NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

HYPERMEDIA ANALYSIS AND NAVIGATION OF DOMAINS

by

Douglas S. Lange

September 1997

Thesis Advisor: Valdis Berzins

Approved for public release; distribution is unlimited.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE **HYPERMEDIA ANALYSIS AND NAVIGATION OF DOMAINS**

5. FUNDING NUMBERS

6. AUTHOR(S) Douglas S. Lange

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Hypermedia systems have been demonstrated to support authoring and reading of mostly static information. Few systems address the needs of analysts deriving information from a continuously changing base of information. Those that do, focus on the existing content and use links primarily for navigation and management. An open hypermedia architecture is proposed for a class of analysis systems where the value added by the analyst is through associating data elements. In such systems, links are the primary form of information being managed.

The architecture developed provides a framework through which hypermedia analysis systems can be generated with little or no code development. Specifically, the model is shown to apply to the domain of software engineering by mapping the analysis portions of a rapid prototyping lifecycle to a schema defined using the framework.

Through the addition of n-ary links and links to links, the architecture provides a closer mapping to the Dexter Hypertext Reference Model than current graph-based models such as the Multimedia Object Retrieval Environment (MORE). Improvements over MORE are also shown in the use of abstraction as a filtering mechanism and through the full involvement of links as being the primary focus of the analysis, query, and filtering functions.

14. SUBJECT TERMS Hypermedia, software requirements, analysis, framework, navigation, object-oriented domains, schema..	15. NUMBER OF PAGES 116
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

HYPERMEDIA ANALYSIS AND NAVIGATION OF DOMAINS

Douglas S. Lange
B.S., University of California at Davis, 1983

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

ABSTRACT

Hypermedia systems have been demonstrated to support authoring and reading of mostly static information. Few systems address the needs of analysts deriving information from a continuously changing base of information. Those that do, focus on the existing content and use links primarily for navigation and management. An open hypermedia architecture is proposed for a class of analysis systems where the value added by the analyst is through associating data elements. In such systems, links are the primary form of information being managed.

The architecture developed provides a framework through which hypermedia analysis systems can be generated with little or no code development. Specifically, the model is shown to apply to the domain of software engineering by mapping the analysis portions of a rapid prototyping lifecycle to a schema defined using the framework.

Through the addition of n-ary links and links to links, the architecture provides a closer mapping to the Dexter Hypertext Reference Model than current graph-based models such as the Multimedia Object Retrieval Environment (MORE). Improvements over MORE are also shown in the use of abstraction as a filtering mechanism and through the full involvement of links as being the primary focus of the analysis, query, and filtering functions.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I want to thank my family for making it possible for me to pursue this goal. My wife Jo Ann took on the extra load at home left behind when I needed to spend extra time working towards this degree. She allowed me to focus on classes and research during this tough two-year period. Alexander and Joshua deserve much credit for being able to cope so well with their father perpetually busy.

I would like to thank my advisor Professor Berzins and second readers Professor Luqi and Dr. Kathleen Fernandes for their advice and ideas. I would also like to thank Professors Berzins, Luqi, Shing, and the rest of the M.S. Software Engineering lecturers for the inspirational classes that inspired us all to think beyond the state of the art.

All of us in the distance learning program from NRaD owe a great deal to Duston Hayward who worked hard to establish this program for us. NRaD management and NPS faculty and management also deserve a lot of credit for supporting what has been a wonderful program for student/employee and employer alike. This has been a wise investment that is already showing dividends.

I. INTRODUCTION

A. A PATTERN OF ANALYSIS

In many domains, analysis is an exercise in making connections between pieces of information. Physicians providing remote consultations through a telemedicine system can be thought of as linking collections of patient encounter information to diagnoses based on their experience. The conditions these diagnoses represent have already been discovered and described in most cases. In addition, treatment protocols have been developed for many of these diagnoses and have been linked to them by medical researchers. The link provided by the consultation can have characteristics. For instance, the link might have attributes that represent the strength of confidence that the physician has in the diagnosis due to the quantity and type of data presented, and the degree to which the physician feels the data match expectations.

Software requirements analysis can be approached in terms of finding elements of the system's domain and their interconnections in the development of or linkage to a problem frame [Jackson, 1995]. These concepts are then linked to solutions that either have been successfully used within the frame before or are newly developed. The individual data elements are not typically being initiated by these analysts, although at times new notions might be introduced. The majority of their efforts create associations between elements that already exist.

Analysis of this form can support command and control environments as well. Actions, orders, and status can be represented through associations. A military planner may choose the best way to engage a hostile unit by evaluating information created by intelligence analysts that has been developed using tools working just as telemedicine tools might function. The results of the planner's analysis are recorded in links between targets, weapon systems, tactics, and particular preconditions (e.g., weather). Further, once presented several alternative plans in this form, a commander may generate an attack order by adding an execution link from the plan to the particular unit that is to carry out the attack. Autonomous agents looking for the creation of such links can set a series of events into place to ensure the order is issued and can evaluate the status of the plan. The

status can be linked to the plan being executed to allow the commander to maintain awareness of the situation.

The figure below shows how a hypergraph can represent the information provided to a commander. Here the targets are being evaluated based on attempts to minimize collateral damage. A report detailing how Target #2 can be attacked while reducing the risk of collateral damage is presented. To order the attack, the commander might connect the target, the tactic, and a platform capable of executing the plan.

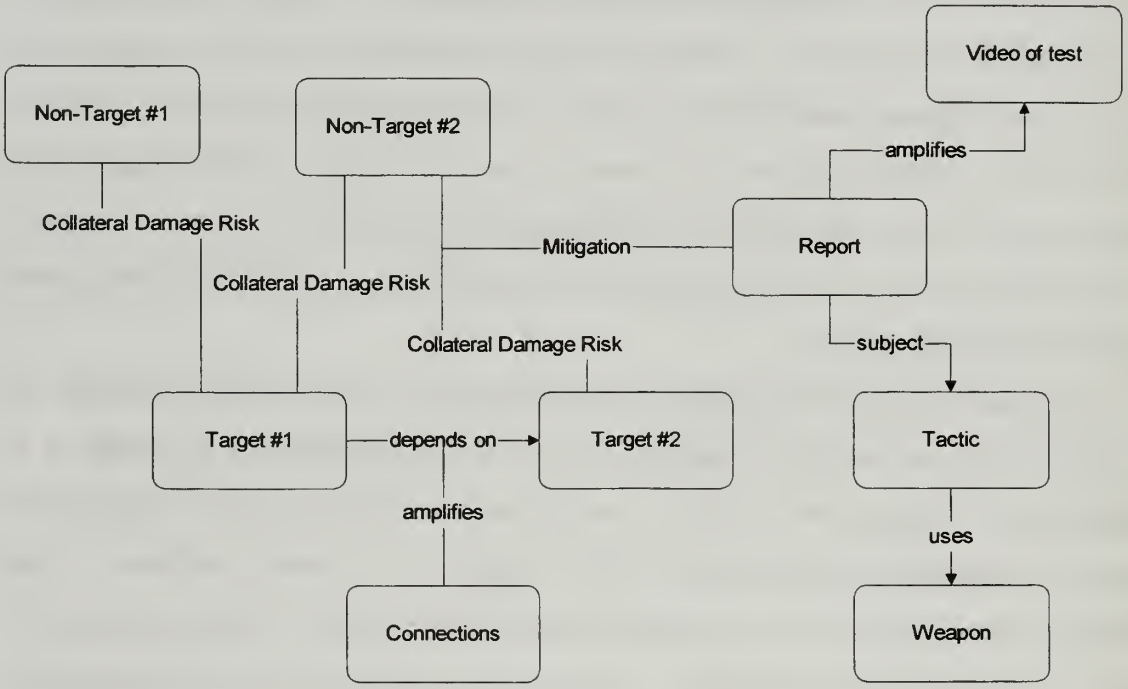

Figure 1.1 Hypergraph of a Plan

The descriptions above fit the criteria for a pattern. They describe a problem that reoccurs and present the core of a solution that can be used multiple times [Alexander et. al., 1977]. In fact, this pattern is illustrated in a famous use case from the memex device proposed as World War II was concluding.

> The owner of the memex, let us say, is interested in the origin and properties of the bow and arrow. Specifically he is studying why the short Turkish bow was apparently superior to the English long bow in the skirmishes of the Crusades. He has dozens of possibly pertinent books and articles in his memex. First he runs through an encyclopedia, finds an interesting but sketchy article, leaves it projected. Next, in a history, he finds another pertinent item, and ties the two together. Thus he goes,

> building a trail of many items. Occasionally he inserts a comment of his own, either linking it into the main trail or joining it by a side trail to a particular item. When it becomes evident that the elastic properties of available materials had a great deal to do with the bow, he branches off on a side trail which takes him through textbooks on elasticity and tables of physical constants. He inserts a page of longhand analysis of his own. Thus he builds a trail of his interest through the maze of materials available to him. [Bush, 1945]

Bush even anticipated the uses of such hypermedia. The future reader is covered under another use case, as are link attributes. One attribute of interest to Bush was the age of a link and the frequency of its use, allowing links that are accessed more recently to be stronger than those that fade with time. Even autonomous agents were envisioned that would take care of mechanical actions based on the user's decisions. [Bush, 1945]

B. USE OF HYPERMEDIA SYSTEMS FOR SOFTWARE EVOLUTION

Information generated in this manner can be captured in the form of a hypergraph model. The hypergraph model of software evolution is an example [Luqi et. al., 1994] [Luqi and Goguen, 1997]. Software analysis is also an exercise in connecting data. Though the end goal is to reuse, create or modify software products, the path taken is one of matching user needs to software requirements, and software requirements to design elements. Throughout this process, analysts create and break associations among pieces of information, and in doing so create new information through these structural modifications. Each change of the structure has a rationale behind it, right or wrong. Each change is an addition to the body of knowledge within the domain.

One mechanism for automating the management of information modeled in this way is through hypermedia systems. Using a hypermedia system that provides both authoring and reading tools, analysts can create information that people and autonomous agents can access. What hypermedia systems provide is the ability to easily work with a wide variety of data while utilizing the powerful information present through the structures created by the connections made among the various data items [Nurnberg, et. al., 1997]. They also allow the user of the information (reader) to access the information

in ways not necessarily planned by the creator of the information (author). In this way, new discoveries can be made from the same body of data as readers with problems different from those envisioned by the author look at the hypertext from a different perspective. [Nielsen, 1990]

Modeling associations is not a unique activity to hypermedia system development. Object-oriented and relational models also place a great deal of importance on how pieces of a domain relate to each other [Rumbaugh et. al., 1991]. Hypermedia stands out in that the links stored constitute much of the information content of the system rather than a means of connection for objects or entities that embody the focus of the system. Object-oriented design methods can prove valuable in developing hypermedia systems once the links are defined as objects themselves [Wiil and Leggett, 1997].

C. CONTRIBUTION

Current hypermedia systems and research focus heavily on the reader of hypermedia information. A number of search and navigation methods and models have been proposed and demonstrated. Yet, these focus on finding and presenting data objects or locating an object currently in focus relative to the entire domain. Filters limit the objects shown in a display, while link presentation is based upon the object subset presented. These approaches downplay the link's contribution to the hypertext system, restricting it to navigation support, rather than treating a link as an atomic unit in the development of a structure that represents knowledge within the system. Additionally, while data objects and links can be described using object-oriented terms of generalization, this attribute is not well used in presenting the information to the reader.

Authors' tool requirements are not well understood beyond the development of online training, journalism, and entertainment products. These systems stress the nonsequential nature of hypermedia [Nielsen, 1990]. The information content of the linkages is an aid to navigation, but also in subtle ways provides information to the user concerning what concepts or actions relate to what others. The dynamic nature of an analyst's work is not typically addressed, and tools for authorship that attempt to build a

domain structure are targeted at software system developers rather than domain analysts as end-users.

For hypermedia systems to be of value to analysts such as those working in software evolution, authorship tools need to have a prominent role. Likewise, links must be the focus of searches and filters since the links are the primary form of information being dealt with by the analyst. Finally, to scale systems and their user interfaces up to handle the complexity of data used by analysts, better use of abstraction needs to be employed.

The main contributions of this thesis are:

- A hypermedia architecture supporting the analysis of software evolution consistent with the Dexter Hypertext Reference Model [Halasz and Schwartz, 1994]. The model presented focuses on the analyst's role in creating hyperlinks and adds value to the links through the addition of attributes that map to the analysis process.
- Models for user interaction with the hypermedia that treat links as equals to data objects in importance to the reader. Searches and filters operate on links as well as data objects and enhance the value of the links through the addition of attributes that map to the analysis process.
- The use of abstraction as a filtering technique to aid the reader in understanding the analysis relevant to the reader's problem. Links made between specialized objects may be of interest to readers who never wish to look at the specialized objects or links themselves but are interested in roaming the domain at a higher level of abstraction.
- The ability for a user to dynamically extend a domain through the addition of object classes and link types. Entirely new hypermedia analysis systems can be generated with little or no coding. The architecture provides a framework for rapidly generating new systems covering new domains.
- An extension of graph-based query and filter methods [Ucarella and Zanzi, 1996] to work with hypergraphs. This addition allows better mapping to the analysis pattern described earlier, and allows a mapping to the Dexter

Hypertext Reference Model, which requires the ability to use n-ary links as well as link-to-link connections.

D. ORGANIZATION OF THESIS

The rest of this thesis is organized as follows. Chapter II provides a review of related research in hypermedia systems and analysis models. For hypermedia, focus is given to works that model the underlying architecture for such systems, authoring tools, and on user interaction approaches. Hypermedia support for analysis that fit the pattern described above are studied to determine what features are essential to provide such support through a hypermedia system. Chapter III describes a mathematical model of an architecture for hypermedia systems that support analytical efforts in general and software engineering in particular. The goal is to provide such support while remaining consistent with the Dexter Hypertext Reference Model. Chapter IV describes author/analyst tools consistent with the model presented in Chapter III. Tools are described for all the types of actions an analyst might make that effect the link storage area structurally. Chapter V provides a model for reader interaction with the hypermedia systems generated from the framework in Chapter III. Filtering, navigation, and searching operations are all considered. As these have been well defined for the data objects themselves, focus is given to operating on the links. The use of abstraction for filtering both links and objects is demonstrated. Chapter VI demonstrates the use of the architecture as applied to the Computer Aided Prototyping System [Luqi and Ketabchi, 1988] in support of software requirements analysis. Conclusions and directions for future study are provided in Chapter VII.

II. TECHNICAL BACKGROUND AND PREVIOUS RESEARCH

A. HYPERMEDIA

1. Structural Computing

a) General

Representing information structurally is widely used though its importance is not always recognized. Researchers in artificial intelligence (AI) have found that different structures constitute different knowledge. Knowledge-based systems often use a network of interrelated units to represent information [Travers, 1989].

Structural computing is a "philosophy of the primacy of structure" [Nurnberg, et. al., 1997]. Most models support the primacy of data objects and allow structures to be built through programming. For some problem domains, the structures represent the knowledge in the system and the data objects merely play a role as a part of the structure. Examples of these domains are argumentation support [Streitz, et. al., 1989], spatial hypertext [Marshall and Shipman, 1993], biological taxonomy and linguistics [Nurnberg, et. al., 1997]. In such domains, certain patterns in structures are developed from primitive elements and are used to represent relevant information [Jordan, et. al., 1989].

b) Hypermedia as a Structural Computing Paradigm

Hypertext and hypermedia are commonly thought of as databases that allow the user to navigate to one item of information from another item. They are conceived of as a network of nodes and links where nodes store information and links provide a cross-reference. Links are thought of by many as merely providing a means of transport that can be activated to allow the user to view the information stored in the connected node. [Shneiderman and Kearsley, 1989]

This definition is limiting. It ignores the information that is created by analysts and that can be represented and stored by linking nodes together. The chemical properties of

water differ from those of unbound hydrogen and oxygen atoms. Similarly, structures created in hypermedia provide different information than the collection of nodes that form a subset of the building blocks used. The results of a domain specific analysis can produce much deeper knowledge than the individual nodes could ever represent.

Hypertext systems through their flexibility offer significant advantages in working with structures. Nodes and links can be used to create any conceivable structures when typing is not imposed [Nelson, 1992]. The addition of types can provide constraints to the structures preventing those that are not acceptable and enriching the information presented by those that are created. Flexible systems that support both ideas have also been proposed and developed. Recognition is given to the idea that in the initial stages of analysis, types may not be known, but that later strong typing of links and nodes may be of use to add additional information [Haake, et. al., 1994]. This flexibility can be realized by defining all object types in a system to be subtypes of a universal type. This can be done in the schemas created through the framework presented in this thesis. When this is done, a process of refinement in which general types are replaced by subtypes that are more specific can be employed.

Structural analysis of hypertext has been used to assist the user in navigation. By looking at metrics regarding the compactness of clusters of nodes, algorithms can suggest where aggregation relationships occur within a hypertext. The analysis ignores the contents of the nodes, as well as any possible typing of nodes and links, focusing instead on the numbers of links coming into and out of the nodes and comparing these numbers with the mean values of these figures for all nodes in the hypertext. Improvements in clustering are made by looking at the strongly connected components and biconnected components. The result of the algorithm is a tree of clusters. [Botafogo and Shneiderman, 1991]

2. Open Hypermedia

From 1987 to 1991 researchers noted that the hypertext systems of the time did not support the needs of collaborative work groups and that they could not be integrated

into the computing environments being used in large enterprises [Halasz, 1987] [Malcolm, et. al., 1991]. Requirements were found for hypermedia systems that were not being addressed. These included:

- Interoperability to access and link information across arbitrary platforms, applications, and data sources.
- Link and node attributes to record who made a link, what the permissions are for the particular link or node and other management information.
- Link anchors that allow attachment to the exact data desired.
- Link types to provide more information about the meaning of a particular link and what functions the link is intended to support.
- Public and private links to support collaborative environments.
- Templates for automating routine analysis tasks.
- Navigational aids that can act as filters and supply powerful querying mechanisms.
- Configuration control so that information of importance in an analysis effort can be developed and managed in hypertext.

To address these requirements, open hypermedia systems evolved. Open hypermedia systems have been defined as those that exhibit the following characteristics [Davis, et. al., 1992]:

- A system that does not impose any markup on the data. By marking up data in order to create hyperlinks, the data is changed making it inaccessible to systems that cannot handle the markup.
- A system that can be integrated with any tool that runs under the host operating system. This can be extended to mean a system that can be integrated with distributed object environments.
- A system in which data and processes may be distributed across a network, and across hardware platforms.
- A system in which there is no artificial distinction between readers and authors. This is quite important for systems supporting analysis.
- A system to which new functionality can be easily added.

Since analysts are both readers and authors of node content and links, these characteristics are vital in a hypermedia system intended to support analysis. Likewise, the ability to link objects without changing them is critical. The information being linked together by the analysts may be coming from other applications and databases with which the hypermedia system has been integrated. These applications will not understand changes imposed on the data in order to support linking. The links must be separated from the content. This is the basic premise of open hypermedia systems and has been demonstrated in research systems such as Microcosm TNG [Goose, et. al., 1995,1997].

3. Rich Hypermedia

Rich Hypermedia systems add attribute information to the structural components of the hypermedia. Links and nodes are classified by type and amplifying meta-data may be added to them [Osterbye and Normark, 1994]. Topological constraints may be imposed on the elements of the hypertext representing the business rules of the domain. An example of a constraint that may be employed in requirements engineering (as described in Chapter VI) restricts *requirement* dependency to *issues* rather than allowing a dependency link to *criticisms*. This implies that user feedback must be analyzed prior to affecting the body of requirements, allowing alternative selection to occur. One major limitation in agents cited in [Shneiderman and Kearsley, 1989] is that they cannot infer information about links other than the fact that there is a connection between two nodes. Rich Hypermedia allows more of this information to be easily found as it is stored as part of the links and nodes themselves.

Rich hypermedia systems are usually found in applications targeting specific domains. Therefore, the meta-data can be deduced through domain analysis. Examples of domains targeted for rich hypermedia to date are software engineering [Osterbye and Normark, 1994] [Garg and Scacchi, 1987] and argumentation [Conklin and Begeman, 1987].

Analysis is often performed in domains where information is known that can be used to create a rich hypertext. The schemas found in the model in Chapter III reflect this

idea. However, analysis often uncovers situations not previously imagined. For this reason the HyperObject Processing Environment (HOPE) model presented in this thesis allows the schema to be enhanced without changes to the code or having a negative effect on the hypermedia. It is also possible for the author of the schema to include an untyped component and untyped link for all others to be derived from. This allows all rules to be broken if necessary, while still providing rich information when possible. This can be accomplished since components and links that act as though they were outside the schema are actually created from classes at a higher level of abstraction. Since class membership can be queried, agents can determine whether or not such a situation has occurred.

4. Spatial Hypermedia

Structure does not necessarily require explicit links between pieces of information in order to exist. Links are the expression of interconnection. However, interconnection can be expressed through spatial structure. One benefit of a spatial representation of structure is that ambiguity or uncertainty can be represented in this fashion. If hypermedia structures are represented by nodes literally being near other nodes for which there exists a relationship, then all nodes are to a greater or lesser extent near to all other nodes. [Marshall and Shipman, 1993]

VIKI is a tool for organizing information. Collections of information can be created by placing nodes within nodes. Nodes are also placed into groupings that correspond to a common relationship, partitioning the information space. [Marshall and Shipman, 1997]

This approach, while useful in the experiments done in [Marshall and Shipman, 1997], would not work well in the complex analytical space of software engineering or command and control. These would require n-dimensional spaces, and the number of relationship types (i.e., n) can be huge. Also, while [Marshall and Shipman, 1997] allowed elements to repetitively appear in the spaces, with a large number of relationships, this could clearly get out of hand in a tool using spatiality for visualization.

5. The Dexter Hypertext Reference Model

The Dexter Hypertext Reference Model (Dexter), was developed to provide a basis for comparison among hypermedia systems, and to provide a common foundation on which to standardize for interoperable exchange [Halasz and Schwartz, 1994]. Many authors make the effort to demonstrate how their architectures map to Dexter. By doing so, hypertext researchers have labeled this as an extremely important model.

The Dexter model defines hypertext systems in terms of a three-layer architecture with well-defined interfaces between the layers. These layers are the *within-component, storage,* and *run-time* layers as shown in the following figure.

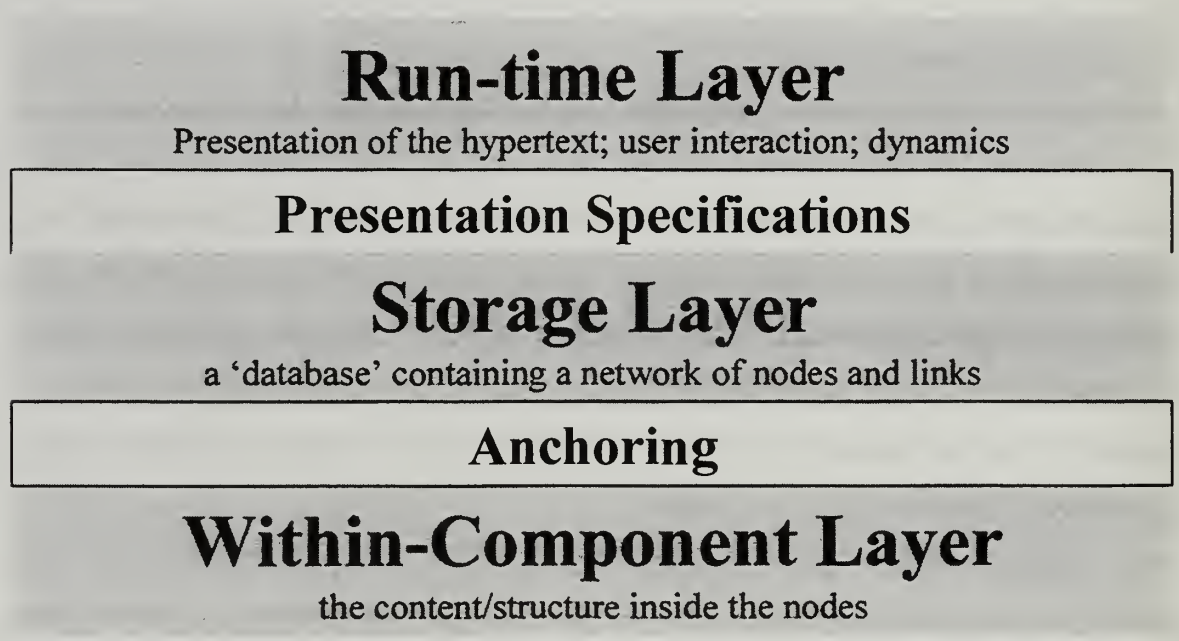

Figure 2.1 Layers of the Dexter Hypertext Reference Model [Halasz and Schwartz, 1994]

The focus of the model is the storage layer and the two interface layers. The storage layer provides the node and link structure that defines hypertext as being a unique architecture. The components of this layer are treated as generic containers. The contents are ignored and are handled in the within-component layer. The interface between the storage and within-component layers is critical to the model however. It is here that the means for addressing locations or items within the content of individual components is handled. This is called anchoring. In the case of pure text components, links are not only available between components, but between spans of characters. Similar spans can be

defined for most media (e.g., spans of video). Likewise, the interface between the storage layer and the run-time layer is important. Through presentation specifications, information about how a component should be presented to a user or to an application is defined. Multiple presentation specifications can be used to allow components to appear in different manners depending on who arrived at that component and by what link. [Halasz and Schwartz, 1994] [Gronbaek and Trigg, 1994]

For hyperobjects as described in this document, it will be shown that anchors and presentation specifications can be handled through the same mechanism. The objects being linked together in the following chapters are not simple text files or other display media. They represent aggregate objects from one or more databases, as in the case of an issue, requirement, or diagnosis. Therefore, spans of characters do not always provide an appropriate anchor for links. Individual fields need to be addressed, and in some cases, spans of characters, or other objects, found within a field. Since information about objects are always received through methods, anchors and presentation specifications are defined through object-oriented design features utilizing methods and their parameters. Design patterns are used that allow each object class to provide a different set of methods to access its data, while not requiring clients to know what type of object is being linked to in advance.

The Dexter reference model while being the most notable model in the field has shortcomings. Despite the stated purpose of allowing interchange of hypertext, limitations have been found. Interchange of hypertext can be accomplished in two basic ways. First, hypertext can be exchanged between hypertext systems. Dexter has been shown to support this approach. A second approach has been called hypermedia-in-the-large. This approach is necessary for large digital libraries or engineering enterprises. It provides a framework for allowing all information in a wide-area network to be utilized with hypermedia applications. Rather than provide a single hypertext system as described for the Dexter model, a link service is provided. Applications that know how to use the link service can access and create hypermedia information. The implication is that a data model for hypertext cannot be assumed. Dexter has not been shown to be successful in supporting hypermedia-in-the-large [Leggett and Schnase, 1994] .

In the HOPE architecture a *linkable* interface definition is used to allow any application or service that implements the interface to provide or use link services. The basic structure of Dexter is preserved, but the underlying implementation assumption of using a firm data model is replaced with the option to have components and links be built by implementing interfaces rather than inheriting implementation [Coad and Mayfield, 1996].

6. Graph Models, Views and Object Retrieval

Unstructured hypermedia models have not been seen as suitable for capturing the knowledge needed for many applications. The result is extra cognitive overhead for both the author and the reader. The author must expend effort in choosing link structures that will prevent the reader from getting lost, and often the reader must choose a path carefully for the same purpose. As a result, graph-based models that incorporate the notion of domain modeling have been introduced. The data modeling aspects, similar to those used by database developers, keep the information consistent with the users frame of reference. Graph models provide a natural way to model associations that form in hypermedia systems. The combination has been used by several researchers. [Lucarella, et. al., 1993]

a) Multimedia Object Retrieval Environment (MORE)

MORE uses just such a graph model [Lucarella and Zanzi, 1996]. The model of this system's architecture is defined using the four-tuple $M = (\Sigma, O, \mathcal{I}, \mathcal{L})$. The conceptual schema Σ is defined by the five-tuple $\Sigma = (C, T, A, \mathcal{P}, \mathcal{H})$ itself. The conceptual schema is the domain definition for the particular multimedia system. The individual elements of these tuples are defined as follows:

- C is a finite set of class names.
- T is a finite set of primitive type names that are built into the multimedia system. $V(t)$ is the set of values associated with the set where $t \in T$.
- A is a finite set of attribute names. Attributes may be simple or complex. Simple attributes have as their domains a type $t \in T$. Complex attributes have

classes $c \in C$ as their domains. The schema defines attributes as being either single or multiple in order to represent the cardinality of the relationships. However this really needs to be a part of the property relationship defined below. Regardless, the mathematics of the sets and graphs does not change for either MORE or HOPE. It is also not necessarily appropriate in all hypermedia systems to provide such a constraint on the attributes. If such constraints are truly needed both models may need to be upgraded.

- $\mathcal{P} \subseteq C \times A \times (C \cup T)$ is the property relationship. If $(c_i, a, c_j) \in \mathcal{P}$, then c_i has the complex attribute a, relating it to class c_j.
- $\mathcal{H} \subseteq C \times C$ is the inheritance partial ordering relationship. If $(c_i, c_j) \in \mathcal{H}$, then c_i is a subclass of c_j.
- O is the set of instances of objects created within the system. Each object belongs to one of the classes $c \in C$.
- $\mathcal{I} \subseteq O \times C$ is the instantiation relationship. If $(o, c) \in \mathcal{I}$ then o is an object of class c.
- $\mathcal{L} \subseteq O \times A \times (O \cup V(T))$ is the link relationship. If $(o_i, a, o_j) \in \mathcal{L}$ then o_i has the attribute a with the value o_j. This is only true if either the classes for the two objects were related with the attribute a, or the property was inherited from ancestral classes.

Graphs are defined for both the schema and the multimedia information system. Each of the graphs is defined as a tuple consisting of a set of nodes and a set of edges. The graph for the schema has for its nodes the union of the classes and types sets ($C \cup T$). The edges are a union of the inheritance and property relationships. Therefore, nodes can be connected by an edge either if they are related by virtue of an attribute or if one class is a subclass of the other. The graph for M has nodes consisting of the object instances of O and the values of the primitive types that are used as attributes for the objects (i.e., $V(T)$). The Edges of M are the link relationship triples defined above.

This model serves as a starting point for the architecture defined in this thesis. HOPE generalizes the graph model to a hypergraph model and provides greater value to the links in the graph structures by making them classes with instances as well. By doing

this, particular instances of a relationship can be created and retrieved, or included and excluded from an overview map of the hypergraph. Additionally, abstraction is used to greater benefit both for component nodes and links by filtering out simple attributes that are not relevant to the level of abstraction being viewed.

b) Nested Context Model

Other models exist that are based on graph structures. These models do not extend to the analysis patterns described in Chapter I as well as that for MORE. The presentation algorithms are however still interesting and can provide ideas for HOPE algorithms. One such model is the Nested Context Model [Casanova, et. al., 1991]. Here, the model is quite simple with a set of nodes (N) and set of links (L). There are two basic types of nodes, context nodes and terminal nodes. Terminal nodes are similar to the primitive types in the MORE model. Context nodes group sets of terminal and context nodes through relationships identified by links. A node may be related to more than one context node, thus the result is a directed graph, where every link is a unidirectional edge and is an attribute of one context node. To have bi-directional links, both nodes would have opposing links defined. This is similar to the property relationship of MORE, however, MORE does not view the link as being contained in a node, nor the feature that context nodes contain the nodes that are connected to them by links.

Therefore, the system is divided into multiple hyperdocuments and a particular node can be contained in any number of them. This creates a hierarchical description of the information where each context node of a particular level is a root, though the root may be a lower level context node in a hierarchy defined by another root and vice versa.

c) HYDESIGN

The Dexter Hypertext Reference model described above requires the ability to handle composite links and components. HYDESIGN [Marmann and Schlageter, 1992] is one of the few models to successfully achieve this. The data model for HYDESIGN follows an object-oriented class hierarchy. The figure below (drawn using Unified Modeling Language [Rational, 1997]) gives a top level view of the model. Atomic nodes compare to the primitive types of MORE and HOPE. One of the primary differences is

that reference nodes may be created allowing more than one component to share the same content. This is not infeasible in the set-based architectures of MORE and the extension to HOPE, but is not currently allowed in either.

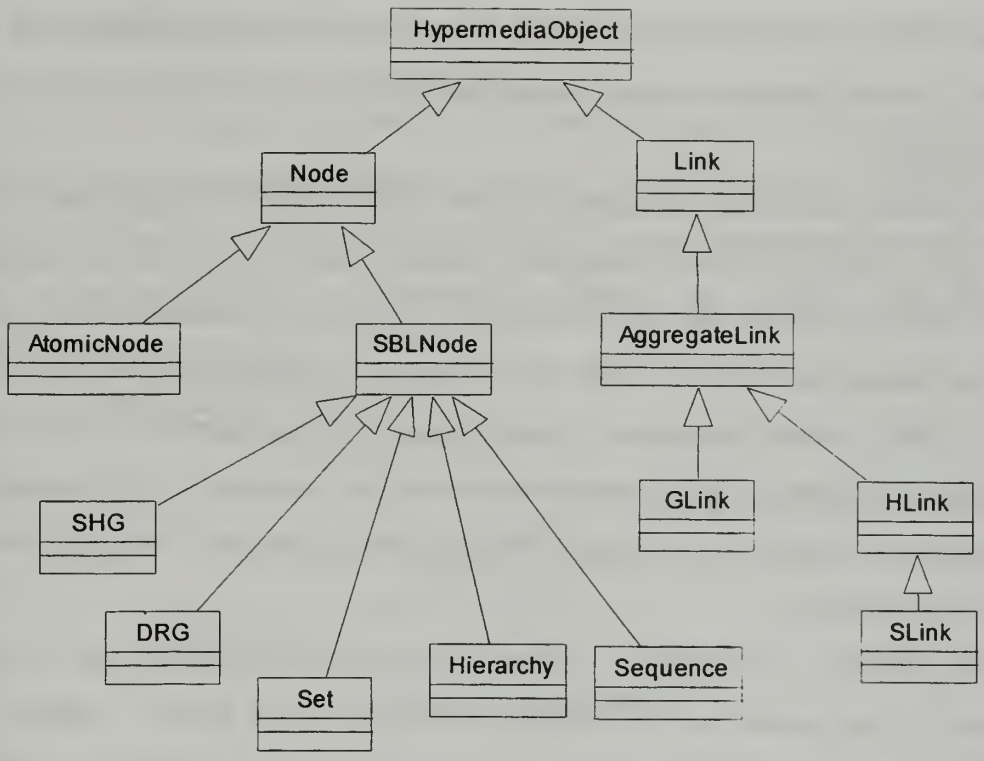

Figure 2.2 HYDESIGN Class Hierarchy [Marmann and Schlageter, 1992]

The most interesting difference among the models from the standpoint of analysis systems is the addition of aggregate links. The extensions to MORE built into HOPE allow for link abstractions. There are three types of aggregate links in HYDESIGN. The first is the g-link or general aggregate link. G-links define directed rooted graphs. When creating g-links, three conditions must be met:

- Source and target node must be of the appropriate types
- There are no link conflicts with any other aggregate links
- The resulting network is a directed rooted graph. All nodes are reachable from the root.

When a g-link is deleted, the system must delete every node that is no longer reachable from the root.

The two other classes of aggregate links are h-links (hierarchical links), and s-links (sequential links). Hierarchical links are similar to g-links except that they form directed rooted trees instead of directed rooted graphs. Sequential links build sequences to define particular paths through multimedia resources. These sequences can then be treated as a single unit. One basic restriction applies to aggregate links. No node may be part of more than one aggregate.

SBL nodes are another interesting extension on the idea of an aggregate. These are high level nodes with all the features of simpler nodes but with three additional properties, that of structure, behavior, and locality. Structure determines the way nodes and links are connected within the SBL node. Behavior determines the way nodes and links react to SBL oriented operations. Locality refers to the way that SBL-nodes can be used to define workspaces or local environments through aggregation. An SBL-node may define a private workspace for an analyst, or a the global workspace (which the private SBL may be a part of).

The addition of aggregates similar to those in HYDESIGN are a likely continuation of the architecture development described in this thesis. However the concept of perspective and adequately determining the level of abstraction to show the user will be more complicated with these features. HYDESIGN uses a different concept for what are termed *views*. Through the *virtual* deletion of certain aggregate nodes and the concentration on particular localities, areas of the hypermedia are not visible to the user. This is a very different approach from perspectives and the abstraction filtering proposed in this thesis. The semantics of the creation and deletion rules of aggregate links must also be compared to actions taken in analysis to determine if this is an appropriate approach for analysis support systems. However it is clear that the use of aggregate links and nodes could be used to model the hypergraph structures described in [Luqi, et. al., 1994] and [Luqi and Goguen, 1997].

B. USE OF HYPERMEDIA TO SUPPORT ANALYSIS

1. Argumentation

One type of analysis system that has been used for research into hypermedia systems has been argumentation support systems. The most notable of these is gIBIS [Conklin and Begeman, 1987]. Here a hypermedia system was established with a firm schema of the sort that can be developed in MORE and HOPE. The schema is represented by the figure below.

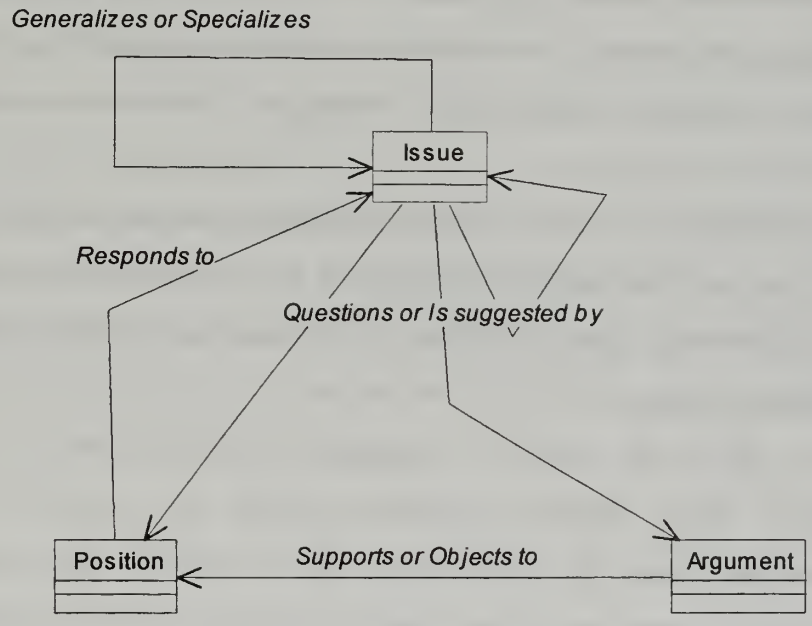

Figure 2.3 Schema for IBIS.

The system (gIBIS) includes capabilities for browsing, context sensitive menus to aid users in making legal moves, searches, and multi-user support. There is no support for perspectives, nor for extending the schema. The idea behind gIBIS is strict adherence to a particular rhetorical style. The lack of perspectives and abstraction however leads to a problem with scaling the system up to many nodes as recognized in [Conklin and Begeman, 1987]. The lack of schema extension was noted in that there were times when a need for a "meta-discussion" was found. New concepts were handled as annotations rather than expanding the schema.

Others have found weaknesses in the approaches taken. The need for guidance to the analyst and the definition of particular structure patterns representing certain types of arguments was noted in [Streitz, et. al., 1989]. The need for particular activity spaces was recognized. The SEPIA system includes:

- *Planning Spaces*: This space is set up to allow coordination of the entire discussion. Posting the initial questions and establishing the structures for participants to respond to is among the capabilities supported in this space.
- *Content Spaces*: This forms space in which the user accesses content concerning the domain and begins to build a semantic structure of subject.
- *Argumentation Spaces*: This serves as a medium for the discussion of the questions to be solved. It is structured as a collection of nodes and links either through a schema as from gIBIS or through a schema representing other argumentation structures.
- *Rhetorical Spaces*: This is a private space that allows the author to structure the arguments that are to be presented in the argumentation spaces. Active applications that assist the author in constructing structures representing particular strategies.

These aspects of SEPIA are useful for requirements engineering and other analysis activities. HOPE allows multiple workspaces through the existence of multiple hypermedia structures within the storage layer of the system. The usefulness of applications to guide the user through the creation of structures is also recognized in the HOPE architecture through the allowance of run-time applications that can manipulate information using storage layer interfaces.

2. Software Engineering

Software engineering has been identified as an activity that can be supported by hypermedia systems at least since 1987, in the Intelligent Software Hypertext System (I-SHYS) [Garg and Scacchi, 1987]. The system is based around the Documents Integration Facility (DIF) which manages software products throughout a systems lifecycle, defining

the products through a series of forms and templates. I-SHYS allows active processes to aid in the software tasks through the knowledge that products relate to each other in a known manner. Agents then can assist analysts in their tasks.

The approach taken in I-SHYS is similar to that described in Chapter VI of this thesis. The application of the HOPE architecture to requirements engineering is done by mapping out a schema that describes the interrelation of artifacts and tasks in the engineering process. Simple tools are provided to allow user to manipulate the structures. However, more intelligent tools can perform task automatically and in support of human effort by exploiting the same interfaces.

I-SHYS also goes further than is demonstrated here in integrating content related tools such as a software engineer might use without a hypertext system. However, the CAPS environment described in Chapter VI, into which this model is meant to be integrated provides the same capabilities.

The I-SHYS work demonstrates the usefulness of a rich hypertext structure that allows autonomous agents to obtain the information needed to provide assistance to the analyst. The capabilities shown in I-SHYS, gIBIS, and SEPIA would not be possible without a schema illustrating the component and association types of the analysis process at hand.

III. THE MODEL

A. CONCEPTUAL SCHEMA

The model presented here for a HyperObject Processing Environment (HOPE) is strongly based on the model used in the Multimedia Object Retrieval Environment (MORE) [Lucarella, et. al., 1993][Lucarella and Zanzi, 1996]. It has been extended to describe a hypergraph rather than a directed graph. This permits a closer mapping to the Dexter model than MORE would achieve. The areas lacking in MORE relate to the requirement in Dexter for bi-directional and n-ary links. In addition, Dexter requires the ability to link to links rather than simply to component nodes.

The primary means by which this is accomplished is through the addition of the *Link* class. Nodes are distinguished as being component nodes (non-link nodes) or link nodes. Links and components become subclasses of a graph's node class. Now edges leaving component nodes do not go directly to other components but are connected to links. Likewise, links can have edges to other links. In this way, multiple components on either end of a link can be connected together and links can be connected to links. This change to the model requires that component nodes of the hypergraph be considered adjacent if there are no component nodes between them. The definition of weakly connected is likewise affected. A new term, "mildly connected" is also introduced as a consequence of this model.

Another change made to the model is that all classes have a common ancestor called *Object*. This is done in the same sense that Java classes are all derived from the class *Object* [Arnold and Gosling, 1996]. This allows all classes, even those not predefined in the schema to be connected using any that does not have a restriction limiting connections to a particular subset of classes. This allows the domain extending capability described in the introduction.

Definition 1. The conceptual schema Σ is defined as the tuple

$$\Sigma = (C, T, A, \mathscr{P}, \mathscr{H}, \mathscr{A}),$$

where:

- C is a finite set of node classes; each class $c \in C$ denotes a structure (in terms of attributes) and an extension (the collection of objects that have this structure). One element of C is the *Link* class. In addition, all classes are derived from a common ancestor known as *Object*. This allows the domain-extending behavior described above and in the introduction.
- T is a finite set of content types. These include the types of the implementation language, and any other classes defined solely in the within content layer. If the type represents content, it is referenced either by a stored query to a database, is referenced in a file-system (e.g., using a path), or references a universal resource locator (URL) in the World Wide Web. The difference between types and classes are that instances of types are not linked to multiple nodes in the hypergraph. They form simple attributes or content of nodes. Each $t \in T$ denotes the type of a primitive object, and $V(t)$ is the set of values possible for that type.
- A is a finite set of attributes. Attributes are functions defined on classes and may be simple or complex. The range of a simple attribute is a basic type $t \in T$ and the range of a complex attribute is a class $c \in C$. Attributes with multiple values (A_m) are distinguished from those that are single-valued (A_s). $A = A_s \cup A_m$.
- $\mathscr{P} \subseteq (C' \times A \times C') \cup (C \times A \times T)$ is the property relationship. C' is defined as follows: $c' \in C' \rightarrow c' \subseteq C$. Here is a significant difference from MORE. The property relation allows n-ary links between subsets of C and unary links between classes and types. If $(c_i', a, c_j') \in \mathscr{P}$, then the elements of c_i' have the attribute a, having as the range the classes in c_j'. Note that this a will be mapped to a particular link class and is therefore explicitly associated with the link class. If more than one class is contained within c' then this indicates that this relationship is only valid when at least one instance of each class is present in the relationship. Later in the definition of a hyperobject multimedia system, it will be shown that as long as the relationship is defined for each class individually, that multiple objects of different classes may participate in such an

n-ary relationship without this requirement. Note also that (c_i', a, c_j') implies a relationship *from* elements of c_i' *to* elements of c_j'. If $(c_j', a, c_i') \in \mathscr{P}$ as well, than the relationship can be considered to be bi-directional. Non-directional relationships are modeled in the same way as bi-directional with the *Link* class associated by $\mathscr{A}$ defined below having an attribute specifying how to characterize the relationship.

- $\mathscr{H} \subseteq C \times C$ is the inheritance partial ordering relationship. If $(c_i, c_j) \in \mathscr{H}$ then the class c_i is a subclass of c_j and inherits attributes.
- $\mathscr{A} \subseteq A \times C^*$ where $C^* \subseteq C$ contains the link class and all of its subclasses. This is an onto relationship such that if $(a, c_i^*) \in \mathscr{A}$ then a is represented by linking classes using the link c_i^*. This set serves to map associations to the link classes that represent them.

Definition 2. Given Σ, the *conceptual schema hypergraph* is a hypergraph

$$H(\Sigma) = (N, E),$$

where:

- $N = C \cup T$ is the set of nodes. Nodes from C contain the components and links of the hypermedia, while those from T contain the primitive types used by component content. For each $c \in C$, where $c \notin C^*$ (i.e., component nodes), there is a rectangular-shaped node labeled c. For each $t \in T$, there is an oval-shaped node labeled t. For those $c \in C^*$ (i.e., link nodes) the node is represented by a line segment. One or more non-link nodes will be connected by edges to either end of the line segment. One or more other link nodes will be connected to a midpoint of the line segment by an edge as well.
- E is the set of edges. For each $(c_i, c_j) \in H$ there is an edge (shown as a bolder line than other edges within the user interface) connecting c_i to c_j. This edge is directed and has an arrow on the side of the parent class. For each $(c_i', a, c_j') \in \mathscr{P}$ there is a link node line segment labeled with a, where the link node is of type c_i^* and $(a, c_i^*) \in \mathscr{A}$. There are edges from each element of c_i' to the first endpoint of the link node, and edges from each element of c_j' to the second. As mentioned in the background research, Dexter based systems can have links

that are to, from, bi-directional, and non-directional. If a link is to, then the arrows of the edges to the first endpoint will point back towards the elements of c_i'. If a link is from, then arrows of the edges connecting to the second endpoint will point to elements of c_j'. Edges connected to bi-directional links will always have arrows pointing back to all of the non-link nodes, and edges to non-directional links will not have any arrows. Edges alone connect nodes to types. Types do not connect to links. These edges are always unidirectional and are presented just as in MORE, as arrows pointing to the type.

These definitions are more complex than those used in MORE [Lucarella and Zanzi, 1996]. The added features implement the characteristics of the hypergraph rather than a directed graph. In addition, these definitions give links an equal status with the classes/nodes representing content. This is a significant improvement over most models in that filtering, searching, and navigation can use links as primary elements rather than merely including them if nodes that they connect are present. As presented earlier, links are the value-added information of the storage layer in a Dexter-based hypermedia system. It is in this realm that analysts work.

B. MULTIMEDIA INFORMATION SYSTEM

The conceptual schema and the conceptual schema graph defined above define the class structure of the hypergraph object model for the system being developed. To obtain the object model, the relationships between classes and their instances as well as between instantiated objects must be defined. The class model illustrates what connections can be made, while the object model shows what links and attribute values are actually present given the state of the user's analysis.

Definition 3. The hyperobject multimedia information system (HOMIS) M is defined as the tuple

$$M = (\Sigma, O, \mathcal{I}, \mathcal{L})$$

where:

- Σ is the conceptual schema defined above.

- O is the set of objects stored into the system.
- $\mathcal{I} \subseteq O \times C$ is the instantiation relationship. Each object $o \in O$ is an instance of a class $c \in C$. Note again that links are members of a class derived from the class *Link* and ultimately from *Object*. This means that actual links are themselves instances and can be treated on an equal footing with other classes within user interaction models.
- $\mathcal{L} \subseteq (O' \times O(A) \times O') \cup (O \times A \times V(T))$ is the link relationship. Paralleling the property relationship, sets of objects (O' is defined as $o' \in O' \rightarrow o' \subseteq O$) can be linked to, from, bi-directional, or without direction. $O(A)$ is the set of instances that represent links. If $(o_i', O(a), o_j') \in \mathcal{L}$, then the objects within o_i' have been linked to the elements in o_j' with a link relating to an instance of the class bound to attribute a in $\mathcal{A}$. If $(o, a, V(t))$ then o has the attribute a with the value $V(t)$. $(o_i', a, o_j') \in \mathcal{L}$ can occur if and only if one of the following conditions holds:
    1. For all elements of o_i' and o_j', the elements are instances of some classes $c_i \in c_i'$ and $c_j \in c_j'$ such that $(c_i', O(a), c_j') \in \mathcal{P}$. This is valid only if c_i' and c_j' are sets with single elements only, or if all elements from these sets have object instances in o_i' and o_j' respectively.
    2. For all elements of o_i' and o_j', the elements are instances of some classes $c_i \in c_i'$ and $c_j \in c_j'$ such that $(c_i, c_k) \in \mathcal{H}$, $c_k \in c_k'$ and $(c_k', O(a), c_j') \in \mathcal{P}$.
    3. For all elements of o_i' and o_j', the elements are instances of some classes $c_i \in c_i'$ and $c_j \in c_j'$ such that $(c_j, c_k) \in \mathcal{H}$, $c_k \in c_k'$ and $(c_i', O(a), c_k') \in \mathcal{P}$.

The conditions above mean that objects can only be related if their classes have been defined as being related in the same way either directly or through the inheritance relationship. These conditions do allow an n-ary relationship to be built up from several unary or smaller n-ary relationships. As stated earlier, these smaller n-ary relationships must be satisfied in whole or they are not valid. These rules can be relaxed by a particular

system through the use of *Object* and *Link* in the schema. This feature can essentially be turned on and off if desired.

Definition 4. Given the HOMIS M, an *instance hypergraph* is a hypergraph

$$H(M) = (N, E),$$

where:

- $N = O \cup V(T)$ is the set of nodes. Nodes represent components in the Dexter model [Halasz and Schwartz, 1994] using rectangles, links using line segments, or values using ovals. These are generated from the schema using the instantiation relationship.
- E is the set of edges connecting component objects to link objects, link objects to link objects, and objects of any sort to values. For each $(o_i', O(a), o_j') \cup (o, a, V(t)) \in \mathscr{L}$ there are edges connecting the nodes, labeled with a and with arrows based on the direction of the relationship. If a link is *to*, then the arrows of the edges to the first endpoint will point back towards the elements of o_i'. If a link is *from*, then arrows of the edges connecting to the second endpoint will point to elements of o_j'. Edges connected to *bi-directional* links will always have arrows pointing back to the non-link nodes, and edges to *non-directional* links will not have any arrows. Edges alone connect nodes to types. Types do not connect to links. These edges are always unidirectional and are presented as arrows pointing to the type.

C. RELATED DEFINITIONS

Decisions on what to present to the user are going to depend on concepts relating whether or not particular nodes are connected in a sub-hypergraph created with a particular criterion. Since links have been defined as nodes in order to achieve n-ary links and link-to-link associations, new definitions with regard to adjacency and connectedness are needed.

Definition 5. Two component (non-link) nodes are adjacent to one another iff, there is a path between them that does not include any other non-link nodes. Two link

nodes are adjacent if they are directly connected by a single edge. Link and component nodes are adjacent to each other if directly connected by a single edge.

This definition allows us to treat the link as though it were an edge in a hypergraph, yet continue to give it the same treatment as other objects with regard to searches, linkages, and filters. It is still useful at times to consider component nodes as being directly related if there are only links between them.

There could be multiple links between them but no intervening nodes. In many cases, this is not a concern and adjacency as defined above is sufficient. However in other cases (e.g., annotation links that may be considered of less importance than other relationships), we may want to differentiate between those separated by a single link and those that can be connected through a variety of links (and perhaps a variety of links and nodes). For this purpose, the following definition is supplied.

Definition 6. Two nodes (link or component) are *mildly connected* iff there is a path from one node to the other, but no path from one to the other can be created without adjacent links.

Definition 7. A hypergraph is *weakly connected* iff by ignoring the direction of the edges, there exists a path from any node to any other node.

D. ARCHITECTURE

1. Storage Layer Composition

The architecture being proposed is a direct translation of the mathematical model described above fitted into the Dexter reference model. In designing an implementation of this architecture, several options have been found, but the architecture remains a description of the sets and tuples of the definitions above. The object model of this architecture is described in the OMT [Rumbaugh et. al., 1991] diagrams below.

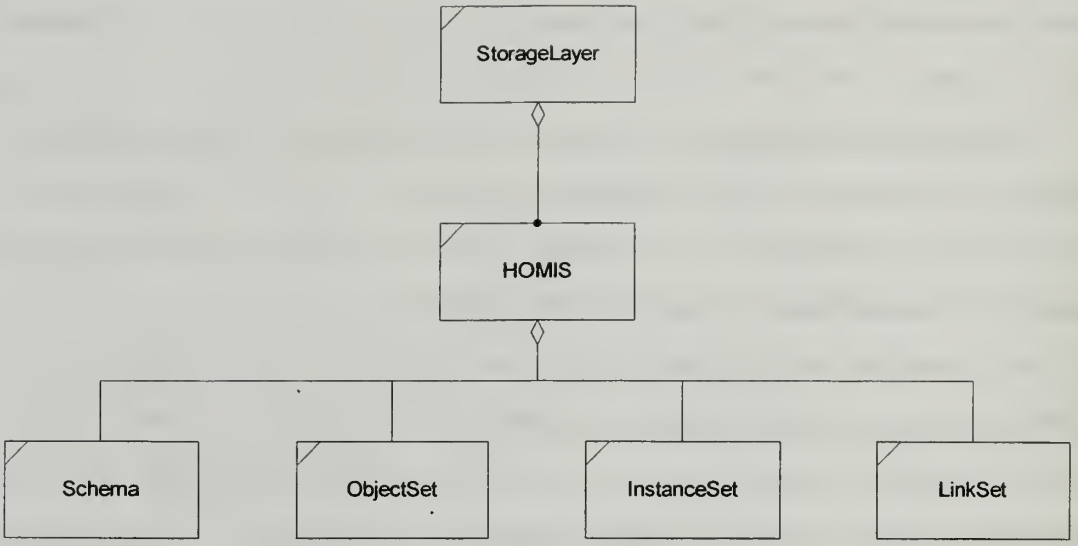

Figure 3.1 Storage Layer Composition

The Dexter storage layer is built as containing zero to many HOMIS instances. Each HOMIS represents $M = (\Sigma, O, \mathcal{I}, \mathcal{L})$. The runtime layer must specify which hypergraph is being dealt with by specifying which HOMIS is being accessed. There can be multiple views of the HOMIS as well as will be described in the model for reader interaction.

The Schema is also composed of sets as described in the model above and the diagram below.

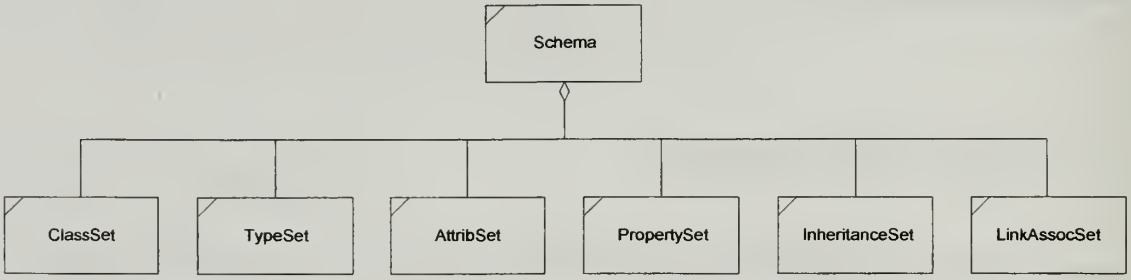

Figure 3.2 Schema Composition

This architecture allows schemas to be built using a drawing tool in the run-time layer. The implication is that new schemas and therefore new HOMIS can be designed without additional code being produced. This is true provided that:

1. TypeSet (T) contains all of the primitive types needed for the new hypermedia system. If not, new code must be written will be to implement the additional

types. This includes the definition of new anchor types that are meaningful to the primitive type. The anchor indicates a method and offset (if any) to get to the content being tied to the component node. As the types are built up, new analysis systems can be rapidly created.

2. No new non-content attributes are required concerning the instances of the components and links. These are part of the tuples stored in the ObjectSet so the definition of the tuple class being used would need to be modified.

The sets themselves need to contain strings, pairs, and triples. The PropertySet contains two types of triples adding some complexity, though this can be implemented easily in polymorphic languages.

Methods for the storage layer are listed in the following table. The initial 13 methods are required to be consistent with the Dexter reference model [Halasz and Schwartz, 1994]. Those methods after are added due to the richer nature of the model for a HOMIS. Not only must inheritance be accounted for, but methods to manipulate the schema are provided.

Storage Layer Methods	
CreateComponent	Creates a new component and adds it to the hypertext.
CreateAtomicComponent	Creates a new atomic component.
CreateLinkComponent	Creates a new link.
CreateCompositeComponent	Creates a new composite component if the atomic components and the composite being created agree with the schema.
CreateNewComponent	Invoked from the run-type layer, and calls one of CreateAtomicComponent, CreateLinkComponent, or CreateCompositeComponent.
DeleteComponent	Deletes a component, ensuring that any links whose specifiers resolve to that component are removed
ModifyComponent	Modifies the non-content attributes (e.g., creation date) of a component while ensuring that its associated information remains unchanged, that its type (atom, link, or composite) remains unchanged, and that the resulting hypertext remains link consistent.
GetComponent	Returns a component from the component's unique identifier.
AttributeValue	Takes a component and an attribute, and returns the value of the attribute. This refers to the primitive type in the HOMIS model. The language between Dexter and MORE conflict.
SetAttributeValue	Takes a component identifier, a value of an attribute, and sets the value of the attribute.

AllAttributes	Returns an enumeration of all the attribute values of the component passed to the method.
LinksToAnchor	Takes an anchor and its containing component, and returns the set of links that refer to the anchor.
LinksTo	Takes a hypertext and a component identifier and returns the identifiers of links resolving to that component.
AddClass	Adds a class to the schema.
DeleteClass	Deletes a class from the schema
AddType	Add a primitive type to the schema
DeleteType	Delete a primitive type from the schema
AddAttribute	Add an attribute to the schema. This can be either an attribute that links two classes together, or an attribute that links a class to a primitive type.
DeleteAttribute	Delete an attribute from the schema
AddProperty	Establishes that a link can exist among from a set of classes to another set of classes using an established attribute. Can also link an class to a primitive type using an established attribute.
DeleteProperty	Deletes a link between class sets or between a class and a type within the schema.
AddInheritance	Add an inheritance relationship from one class to another.
DeleteInheritance	Delete an inheritance relationship.
ListHOMIS	Provide an enumeration of hypermedia systems kept within the storage layer.

Table 3.1 Storage Layer Methods

2. Within Content Layer

The within content layer is structured as a collection of primitives. These primitive types include those typically implemented in programming languages or their libraries (e.g., integer or string), and those that are unique to multimedia systems (e.g., mpeg or hypertext markup language). Objects for each of these primitives are written and containers are built for each. These containers and objects can understand the anchors being passed to them. When an object in this layer is passed an anchor, it returns an object of the expected type. This object can be utilized by presentation specification methods to display the contents for the node of a hypergraph.

E. BENEFIT SUMMARY FOR THE HOMIS MODEL

There are several benefits of the HOMIS model over existing graph-based hypermedia system models.

1. Closer Mapping to Dexter

The HOMIS model includes features that map to the Dexter Hypertext Reference Model [Halasz and Schwartz, 1994] that are not found in existing graph-based hypermedia system models (e.g., MORE [Lucarella and Zanzi, 1996]).

Dexter calls for n-ary links, which are not possible in simple graph-based models. By extending the model to a hypergraph, n-ary links become feasible. Dexter's links must also be able to be attached directly to other links with no intervening components. This is contrary to the mathematics of graph-based models. However, by allowing links to be nodes themselves this becomes possible. HOMIS distinguishes between link nodes and component nodes in order to differentiate between content and associations to preserve the hypermedia feature of the architecture.

The final improvement is the decoupling of content from the *storage layer* or hypermedia services themselves. MORE and others treat content the same as other attributes of component nodes. This is a common feature of object and graph based hypermedia systems that use data objects linked together rather than more traditional media objects (text, video, and audio). The result is a tight coupling between the content of the nodes and the hypermedia structures being created by the authors or analysts. Dexter is an open hypermedia architecture and thus allows attributes of components and links to be separated from the content being represented by the components. In this way immutable objects held within the *within content* layer of Dexter are not affected by the analyst's work. The analyst is truly adding valuable information exclusively through the authoring of links.

2. Less Discrimination Against Links

If authors of hypermedia are adding value through the creation of links and not through the modification of the content referenced by the nodes, why is it that most search, navigation, and filtering methods act primarily against the attributes and content of components. By creating *Link* classes that are treated in a similar manner as component nodes, links are placed level with content in importance and in visibility. This model will assist in developing filtering, navigation, and search tools that focus on the links. Links can now have attributes and can be manipulated independently of the nodes they connect. However, the model still preserves the meaning of a link by tying links to associations that must exist between the nodes they connect. Similarly, research into structural computing may benefit from this model as the structures created through combinations of links and components are of more interest than the content referenced by nodes.

3. Richer Modeling of Domains

Some domains are more easily modeled using hypergraphs than graphs. This is true of software evolution [Luqi, et. al., 1994] and military planning. The lack of n-ary links and link-to-link connections does not prohibit the modeling of these undertakings. By adding placeholder components absent of meaningful content but connected to the necessary nodes, one can model the same information. These placeholder components must have rules associated with them that indicate how the relationships that they represent work. However, this approach adds complexity to the view provided to the user, and since under older models, components are the primary objects, adds clutter rather than meaning. By extending the graph model of hypertext to a hypergraph model, these domains are more cleanly and expressively defined.

4. Use as a Framework

The HOMIS model and architecture provides a means of building a storage layer that does not need new code for each instance of a hypermedia system. Users can enter

schemas to represent the analysis to be done. Only if new primitive types are employed must code be written. This is necessary since the storage layer must communicate with the within-content layer to retrieve data of the appropriate type. Presentation specifications for the type must also be created to provide methods for interacting with the data.

IV. ANALYST TOOLS

This chapter looks at the run-time layer operations affecting a HyperObject Multimedia Information System (HOMIS). The HOMIS is manipulated through calls to the storage layer as defined in the last chapter. The following operations are required by the Dexter Hypertext reference model [Halasz and Schwartz, 1994].

Run-Time Layer Operations	
OpenSession	Creates a session for a given hypermedia.
OpenComponent	Opens a set of new instantiations on a given set of components.
PresentComponent	Takes a specifier and a presentation specification and creates an instantiation for the associated component.
FollowLink	Uses openComponents to present any components referred to by the "TO" specifiers of any links with anchors represented by a given link marker.
NewComponent	Opens a new instantiation on a newly created component.
UnPresent	Removes an instantiation
EditInstantiation	Edits an instantiation. A call to realizeEdits is required to save the changes.
RelizeEdits	Saves an instantiation's editing changes to the corresponding component by calling ModifyComponent.
DeleteComponent	Deletes the component associated with a given instantiation. Also removes any other instantiations for that component.
CloseSession	Closes a given session. Note that by default, pending changes to instantiations are not saved.

Table 4.1 Dexter Run-time operations

As with the storage layer, the HOPE model allows some operations in addition to these minimum requirements. While the model behind MORE [Lucarella and Zanzi, 1996] is geared towards the reading of a static hypermedia, the HOPE model is intended to support dynamic analysis. The operations described below represent the required set and those extensions believed needed to support analysts' use of hypermedia. Use cases supporting these operations defined for the software engineering domain are found in Appendix B.

A. OPEN AND CLOSE A SESSION

The HyperObject Processing Environment (HOPE) can store multiple HyperObject Multimedia Systems (HOMIS) within its storage layer. Each HOMIS represents the information described by a single hypergraph. Therefore, the openSession operation of Dexter is implemented such that the particular HOMIS to be used is chosen from the storage layer. This can be done by providing a selection of HOMIS discovered using the listHOMIS method. A new session can also be created that defines a new HOMIS.

Closing a session always prompts a user to save the modifications made to the HOMIS before ending. This operation calls the relizeEdits operation required of Dexter based systems.

B. CREATE AND MODIFY THE SCHEMA

This is not a capability required by the Dexter reference model. While typed links and components are supported, they are not required nor is there a requirement for a framework to be built that allows new schemas to be created without resorting to coding. However, manipulating the schema sets are no different than manipulating the instance sets. Actions in the graph view of the schema correspond to changes in the sets in the storage layer. Additionally, manipulating the schema for purposes of setting perspectives is merely an extension of the capability to draw and edit a schema. Many of the interactions that the user performs within HOPE are with the schema, making this an important run-time tool.

C. ADD OR EDIT A COMPONENT NODE

Not all component information will already exist. Though the premise of the architecture is that much of this information comes from outside the hypermedia architecture, some is clearly initiated by users of HOPE. In the case of requirements analysis discussed in the use cases and Chapter VI, users must enter criticisms and project

managers must enter demonstration plans. Editing is allowed only sometimes. For the most part HOPE considers component node content as immutable. Anchor logic in the within content layer controls whether write methods are available. Likewise, anchors can force new nodes to be created when information changes from underneath the system. This again stems from the notion that much of the information is controlled by other systems with which HOPE must integrate. Presentation specifications for components control the behavior concerning how the information is presented to and edited by the user. If the component's content is immutable, then the run-time application must actually "spawn" a new instance through the storage layer, making intelligent decisions as to what links to carry forward to the new version, and what type of links to make between the old and new versions.

D. LINK NODES TOGETHER (INCLUDES DELETING OR MODIFYING A LINK)

This is the basic representation of the analysts' effort. Run-time tools need to be in place to allow the user to graphically or through searches, create links between components, between links, and between components and links. Basic tools can graphically allow link creation. The tool can check to see what links are allowed between two items selected by querying the properties set of the HOMIS' schema ($\mathscr{P}$). Task specific applications can help make links based on the actions taken by participants in the activity.

An example is a criticism entry tool for requirements engineering (see Chapter VI). Such a tool can create criticism content, create a new component node, then create a link to the component node representing a user who has the same name or email address of the person who submitted the criticism. This can be done either with direct usage of a HOPE based system, or by an autonomous agent that acts upon receiving criticism content by electronic mail from the user.

E. VIEW AND FILTER THE HYPERGRAPH

This subject is handled in depth in Chapter V. In Chapter VI it is shown that in any real analysis can quickly result in a hypergraph that cannot be displayed. Filtering the screen to see the information of use is important. Since a HOMIS has a schema as well as instance sets, the hypergraph shown to the user can be filtered both through choosing the types of content of interest as well as particular values of interest.

F. EXAMINE A COMPONENT OR LINK NODE

Examining a node allows one to view the contents or attributes of the node. In this model, such content is in the form of "simple" attributes, associations with primitive data elements found in the within content layer through anchors. Therefore, there is no one element that makes up the content of a node, but there may be any number of such attributes. Therefore, users examine attributes through selection of the attribute nodes in the hypergraph. The attributes shown are based on the perspective pattern being viewed. Instances of classes that extend other classes only display the attributes inherited from the level of abstraction selected for the particular perspective(this is described in more detail in Chapter V).

G. FOLLOW A LINK

Any run-time application that is working with a node can query the storage layer with the FollowLink method. If c is the current node and $c \in X$, the storage layer returns the set of links of the form

$$(X, a, Y)$$

where (X, a, Y) is an element of $\mathscr{L}$. Given that the HOPE model includes the concept of perspective, $\mathscr{L}$ can be filtered to include only those links that are present in an instance set $s \in S$ of the perspective $P(\pi, S)$, described in Chapter V.

Since a link node associates sets of nodes (where a node can be either a component or link), the application must choose which node to travel to. This can be as

simple as presenting the list of nodes to a user for choice. Likewise, the application might be written to treat the set as an enumeration and provide all of the nodes to requestor either sequentially or simultaneously. Finally, the application may have an algorithm by which a subset of the nodes are chosen.

H. FIND COMPONENT AND LINK NODES

Searching for particular elements of a database is a common activity. As the hypermedia base is a form of database [Wiil and Leggett, 1997], it is not surprising that this would be common in HOPE as well. Most hypermedia systems support searching for component nodes. By treating links in the same manner as components, links can be the subject of searches as well. Link classes have instances, which can have attribute values, making them suitable targets for searches.

Both kinds of nodes are typed in HOPE. As a result, searches look for instances of a particular class, associated with either:

- Simple attributes of a particular primitive type holding a particular value
- Complex attributes that have particular simple attribute values associated with them.

It is also possible to fashion query capabilities that will use the navigation capabilities described above (FollowLink) to move from node to node that at least satisfy some particular navigation criteria in search of a target node that meets the actual search criteria.

I. DISTANCE AND COMPLEXITY MEASUREMENTS

These capabilities are not usually discussed relative to hypermedia architectures. The need for this capability is suggested in the use cases on alternatives analysis in Appendix B. Two mechanical methods of comparing suggested alternative solutions to issues (as described in the schema for CAPS in Chapter VI) are presented in [Ibrahim, 1996]. The first measures the scope of a proposed change by counting the number of

modules affected by the change, directly and indirectly. In a HOPE based system, this is done by travelling the affects links from an issue and counting the number of component nodes involved. This involves using the FollowLink interface to the storage layer. The second measurement is done by looking at the maximum distance that a module resides away from the issue being resolved. The method of implementing this measurement is the same as for the previous one. Following links, an application keeps count of the links traveled until a module no longer affects any others.

V. INFORMATION RETRIEVAL

A. READER'S GOAL

The reader of hypermedia information has one basic goal in mind: to find the information needed to accomplish a particular task. This basic goal is independent of the nature of the reader. The reader could be human or an autonomous agent. The reader could be searching for information in order to make a decision that will only impact entities outside the systems automation boundary. Alternatively, the reader could be an analyst who once finding the information sought, will link it to another item, changing the contents of the system.

Regardless, readers of hypermedia have two basic modes of information retrieval available to them, search and browsing. Often users will accomplish their tasks using some combination of these two methods. [Nielsen, 1990]

Each of these methods has advantages and disadvantages. In order to be efficient, searching requires that a user understand:

- a query language,
- the structure of the information held within the system.
- and, the particular goals of the query.

The first two of these requirements poses the difficulty that the user is required to know too much about the implementation of the system. However, searching using a query can provide an immediate answer when a well-structured query is properly evaluated [Lucarella and Zanzi, 1996].

Browsing allows the user to view the contents of a component, then using links, decide upon another component to view. If the authors have created links that match the thought process needed to extract the information desired, this can proceed efficiently. The reader has no need for knowledge of the information structure or a query language. The user does not even need to know precisely what is being sought, merely be able to recognize when it has been found. However, if the links are not well suited to the problem being addressed, disorientation of the user results [Marmann and Schlageter, 1992].

B. OVERVIEW DIAGRAMS

Overview graphics can perform multiple functions. Since hypermedia systems are often used through browsing and navigation, overview graphics can provide a sense of what information is nearby and place the user's focus in context to the entire information base. In some systems, the reader can acquire information directly through the overview diagram. [Nielsen, 1995] [Tochtermann and Dittrich, 1992]

HOPE based systems are intended to provide this capability. Since the analytical information being represented is provided by the linking of nodes together, seeing what information is connected provides a great deal of the information in the system. The overview diagram provides a view of the analysis being performed. It also provides the basis for graphical search capabilities described below.

1. Simple Hypergraph View

Any graph based overview diagram is possible using the HOPE architecture. The primary view that is provided in the trial implementation is a simple display of the hypergraph. Later in this section the concept of perspective and filtering are discussed to show how this view is made more useful. At its simplest however, the view is provided a set of object instances and a set of links. From this a hypergraph is drawn following the convention in [Lucarella and Zanzi, 1996].

All component nodes (representing the elements of the set *ObjectSet* or *O* in the HOMIS) are drawn as rectangles. Attributes of the objects that refer to primitive types (or members of *T*) are drawn as ellipses. An edge with an arrowhead is drawn from the object to the node representing the value of the type. The edge is labeled with the attribute name. Links from one object to another are drawn as line segments. Each line segment has three attachment points, two on the ends and one in the middle. The point in the middle is for attaching edges to other links. The endpoints are for attaching edges to component nodes (representing the objects). Arrowheads are provided where direction is defined. An example diagram follows:

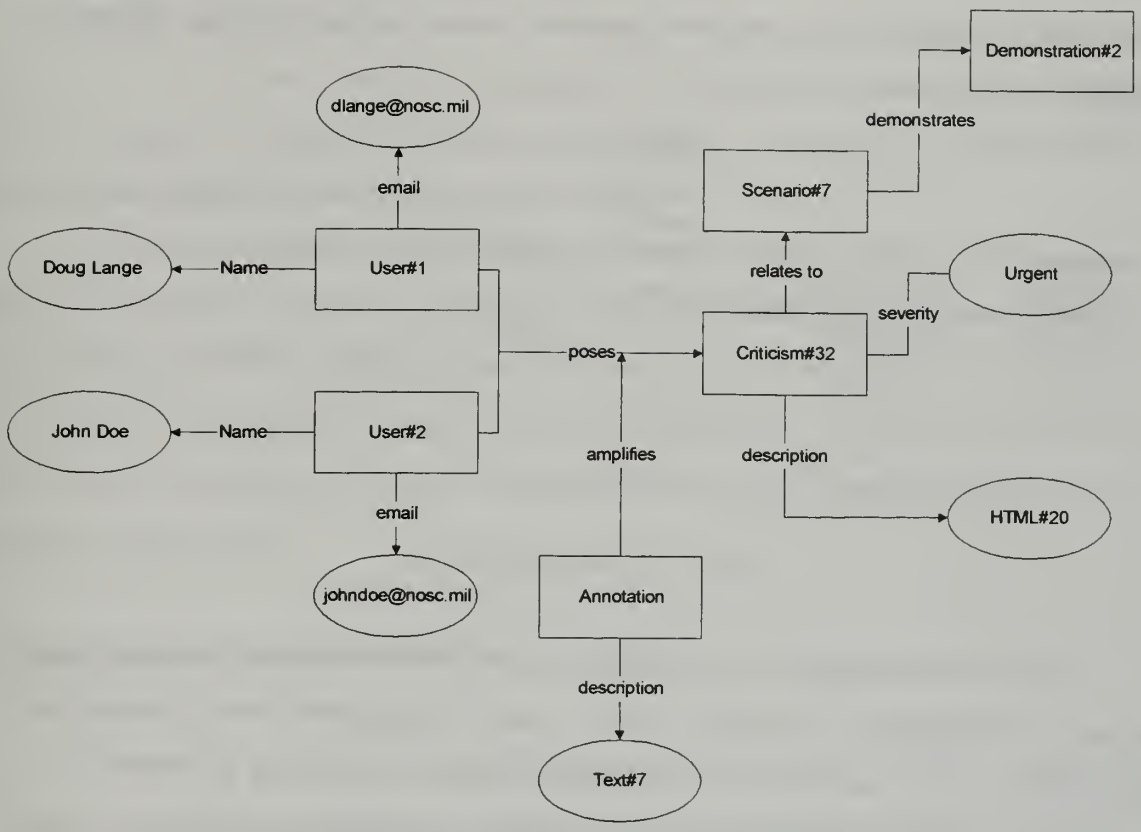

Figure 5.1 Overview Diagram Example

Note that when the values *V(t)* are simple enough to show within the ellipse they are, while an identifier is shown otherwise.

2. Graphical Queries

In MORE and HOPE, to specify classes of objects, the user interacts with the schema to select component or link nodes of interest. HOPE differs from MORE in that the level of abstraction can be chosen by the user. Attributes relating to the level of abstraction are the only ones displayed. Those from subclasses are hidden.

To indicate instances that contain a particular value, the user again interacts with the schema (or more accurately with the perspective pattern described below). The user selects the simple attribute of the class and is given an input window to enter a value for the primitive type. Objects where *V(t)* equals that value can then be retrieved . This is shown in the following figure, where the criticism analysis perspective from Chapter VI is

being used to retrieve only criticisms posed by a user with the email address of *dlange@nosc.mil.*

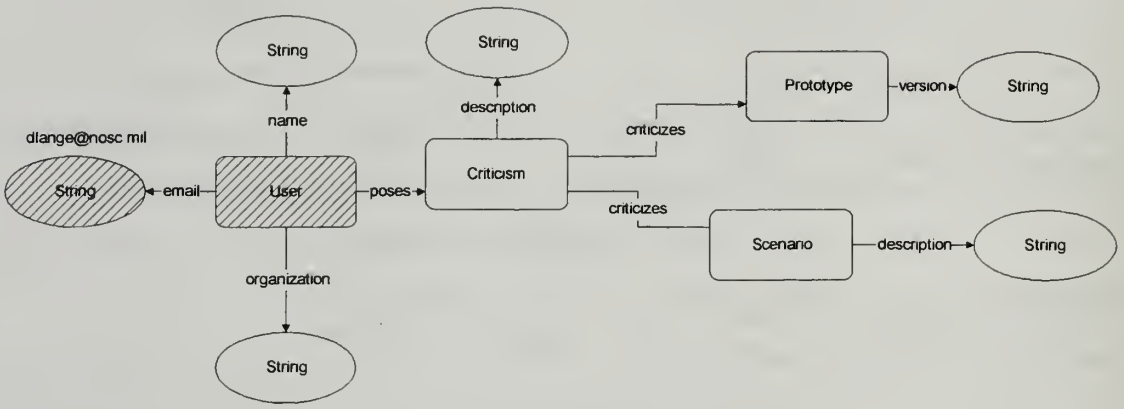

Figure 5.2 Graphical Query

Full Boolean statements can be described in this manner and used for simple object retrieval or to set filters as described below. None of this differs from [Lucarella and Zanzi, 1996]. The basic mode of information retrieval supported by MORE is to successively filter the information in this way until the information of interest is easily visible in the resulting perspective view. In this way, the user does not need to formulate the entire query based on knowledge of the schema. Results are accomplished by graphically filtering the information. It is easier for a user to recognize when successive filtering has resulted in the information required than it is for the user to formulate a suitable query without in depth knowledge of the entire schema.

However, differences do exist between the MORE and HOPE models. The HOPE model allows two additional graphical capabilities. The first is filtering through abstraction. Since users and sponsors are both examples of people, they share certain attributes. Other attributes are unique to the specific type of people. A query where the class people is selected rather than users and sponsors would provide all of the same instances, but would only offer the common attributes for filtering. Where only users are truly of interest, this class may be chosen, hiding all of the sponsors and showing all of the users' attributes (those inherited as well as those that are unique). It is proposed here, but not proven, that this will enhance the ability of the user to find the information desired. A study of this would need to be conducted to determine if this is true. Any query that can

be described through standard query languages can be created using perspective interaction. If a mistake is made in selecting levels of abstraction early in the successive filtering process, the query desired may not be possible. The user must go back to that stage and set a different perspective pattern or filter.

The second difference is that links are classes in HOPE just as are component nodes. The analyst may select links for the purpose of establishing a perspective pattern. Likewise, links can have simple attributes and these may be used to filter the information. Given that HOPE is intended to provide service to analysts who are more interested in the association of information than the raw information itself, it is believed that this will be a significant improvement.

C. PERSPECTIVE AND FILTERING

The concept of perspective is defined in [Lucarella and Zanzi, 1996] as a means to focus the overview diagram on the component nodes of interest to the reader. Perspective is a form of data abstraction that provides the user the ability to select a particular portion of the schema that is of interest. Only object instances that conform to this schema subset are displayed to the user. Perspectives can be predefined and stored for later use. A software requirements analyst can use a perspective that focuses on user criticisms and the issues they represent (see schema in Chapter VI) while ignoring the connections between issues, software requirements, and design elements until ready to address issue solutions. The perspective is built by selecting nodes on the graph in MORE or in HOPE by selecting either nodes or links.

Two changes are introduced in the perspective definition for HOPE.

1. Abstraction is preserved in the perspective. In MORE, selection of an ancestral class implies selection of the descendants. This does not allow the user to set abstraction at appropriate levels for different pieces of information. In HOPE, ancestral class selection allows the objects of descendent classes to be displayed, but the attributes shown are based only on those inherited from

the selected class. In essence, all objects are treated as though they were direct instances of the chosen class.

2. The focus on component nodes is supplemented with a perspective based on the relationship function. In MORE, nodes are selected and links are included if they are associated with the nodes. In an analysis system, the perspective is just as likely to be based on the associations themselves. In this case, nodes are brought in if they are relevant to the associations.

1. Definition of Perspective

Definition 8. Given a HyperObject Multimedia Information System (HOMIS), a perspective P is defined as $P(\pi, S)$, where:

- π is the perspective pattern. The pattern is a weakly connected sub-hypergraph of the schema Σ all of whose link nodes have edges attached to both endpoints. $N(\pi) \subseteq N(\Sigma)$ denotes the subset of the schema nodes (classes, types, and links) included in the perspective. $E(\pi) \subseteq E(\Sigma)$ denotes the set of edges associated with the perspective.
- S is the set of object hypergraphs generated by the perspective hypergraph π through the instantiation relationship. Given $s \in S$, each node (component or link) $o \in N(s)$ is an instance of the corresponding node $c \in N(\pi)$ or $(c, c') \in \mathscr{H}'$ where $\mathscr{H}'$ is the transitive closure of $\mathscr{H}$ and $c' \in N(\pi)$. If $c \in N(\pi)$ then the type $t \in N(\pi)$ where $(c, a, t) \in \mathscr{P}$. Otherwise, if c' is the member of $N(\pi)$, and objects of c are used only because of inheritance, then $t \in N(\pi)$ only if $(c', a, t) \in \mathscr{P}$. The edge $(o_i, a, o_j) \in E(s)$ *iff* the edge $(c_i, a, c_j) \in E(\pi)$.

Since links are given the same standing as component nodes in HOPE, the link associations can be a selection criteria for perspectives as easily as the components can. The second portion of the definition hides primitive type attributes that are found in descendants of a selected class. This allows the objects of descendent classes to be treated as instances of the ancestor that was selected for the perspective.

2. Filtering

The purpose of a filter is to refine the view provided by a perspective to a subset of pattern instances. The user is allowed to choose which of the object instances (by attribute values) should be shown. The display is already limited to particular node (both component and link) types based on the perspective. As in previous sections, the basis for these definitions are [Lucarella and Zanzi, 1996]. The definitions are modified to account for the changes made to the model in support of hypergraphs, the Dexter Hypertext Reference Mode, and the need for more focus on links in support of analysis.

Definition 9: Given a perspective $P(\pi, S)$, a *filter* F is defined in terms of a set of selection conditions $\{C_1, \ldots, C_n\}$ over the pattern. Let a_i be an attribute of type t pertaining to a node (class) n_i in the pattern π; then C_1 represents a selection condition over the actual values of the corresponding object instances. The selection condition is a Boolean combination of simple expressions comparing two values a_i and a_j.

The user interface for setting such a filter allows the user to "click" on a node, then on the attribute to be tested. A window asking for a particular value and providing valid comparison operator choices is presented for the user to fill in.

Consistent with the goals throughout this thesis, the objects being evaluated can be link objects as well as component objects. Link objects can also have attributes that can be compared. The semantics of such mixed queries work as well. Since the component and link nodes are tied (or associated) to each other by edges in the graph, they are treated just as all node objects are in [Lucarella and Zanzi, 1996]. Likewise, abstraction is supported by this approach. The only attributes that are made available to the user are those inherited from the classes chosen for the perspective. Therefore, the level of detail is maintained through the filter operation.

Definition 10: Given a perspective $P(\pi, S)$ and a filter F defined over P, a selection operation σ returns a subset $R \subseteq S$ of pattern instances matching the filter: A pattern instance s matches the filter iff it satisfies all of the conditions C.

Definition 11: Perspectives are *compatible* if they have the same pattern but different instance sets.

Compatible perspectives are used primarily in perspective operations. The primary example of this is in the overlay operation defined later. By knowing that the pattern (or subset of the schema graph) is identical, the operation can combine object instances that pass through different filters. This serves as a union operation on the instances alone and can only be done if the patterns are the same and therefore from compatible perspectives.

The concept of perspective requires further research. The filters defined above only take into account the values of attributes with primitive types. However, attributes connecting to components and mildly connected to components containing particular content could work. In addition, filters that are based on attributes connecting targeted structures of sub-hypergraphs could be defined as well.

Graphical searches work in the same way as filtering. A search is merely a filter that intends to allow through only those elements relating to the search criteria. In this case it is not for the purpose of reducing clutter in an overview diagram, but for returning particular components or links.

3. Other Operations on Perspectives

Perspectives have the same form as schemas and are therefore visible through the same viewers. Likewise, perspectives can be further refined using perspective selection since as a legitimate schema, the operations on a schema are valid on the perspective. The following definitions provide the operations defined in [Lucarella and Zanzi, 1996] using the modified structures that support hypergraphs.

Definition 12. Let $P1(\pi_1, S_1)$ *and* $P2(\pi_2, S_2)$ be two perspectives with $\pi_1 \neq \pi_2$ and $N(\pi_1) \cap N(\pi_2) \neq 0$. A *composition* operation over the set of nodes $N \subseteq N(\pi_1) \cap N(\pi_2)$ generates the perspective $P(\pi, S) = composition(P_1, P_2)$ where:

- π is the pattern that results from taking the unions of the nodes and the edges of the individual patterns. Therefore, $N(\pi) = N(\pi_1) \cup N(\pi_2)$ and $E(\pi) = E(\pi_1) \cup E(\pi_2)$.
- S is the set of instances of the pattern π, created by finding instance sets such that if a node appears in both patterns the refer to the same object instance.

For all of the classes $N' = N(\pi_1) \cap N(\pi_2)$, two instances can only be part of the composition if they share the same object instances.

The differences between the two models shows up here as in previous definitions. Nodes include "link nodes" in the HOPE model. Therefore, composition can occur on the relationships between component objects as well as the component objects themselves. The result is that composition continues to work with the links having a more significant role. Hypergraphs are also supported in this way. This also shows that having instances of links does not interfere with processing the hypermedia.

Definition 13. Let $P_1(\pi_1, S_1)$ *and* $P_2(\pi_2, S_2)$ be two compatible perspectives. This means that $\pi_1 = \pi_2$, but $S_1 \neq S_2$. An operation *overlay* $(P_1, P_2) = P(\pi, S)$ where:

- $\pi = \pi_1 = \pi_2$
- $S = \{ s \mid s \in S_1 \vee s \in S_2 \}$

This is essentially the union of all instances between two perspectives with different filters but the same pattern. Again, the inclusion of association nodes to support the hypergraph and treating links similarly to components, does not pose any problems to the mathematics originally created for perspectives on simple hypermedia graphs.

The next three definitions provide operations that are important characteristics of hypermedia systems, whether or not an overview graphic is provided. Here browsing is done from the perspective overview in order to determine what objects are available from a particular class and present through a particular filter. Navigation on the other hand is conducted from an instance to another instance that exists within a particular perspective. Two differences from the graph model in [Lucarella and Zanzi, 1996] are important to note. First, browsing and navigation can return link objects in addition to component objects. This may not be intuitive and run-time tools may need to be developed that behave differently. If a user "clicks" on a link node in the perspective view of the schema and is given a list of link instances, the run-time tools will have to decide what to do when the user selects one of elements in the list. Different tools legitimately could provide different results. Options include:

- Show a graph including the link and all of the components connected to that link. Mildly connected components and the links that create the paths could be shown as well.
- Allow the user to *navigate* (also defined below) in the direction of the link and be presented with a list of components that could be visited. The revised definition of adjacency presented in Chapter III becomes of interest as the tool must decide what to display to the user in the case of a link-to-link situation.

Second, the navigation operation must consider the revised definition of adjacency and the presence of n-ary links. Since links-to-links are possible, adjacent component nodes are the targets of navigation, not the links that connect them. This may seem like an abandonment of focus on links and a reversion to primacy of node content, but it is not. This is the one operation for which links truly are merely a path to follow, as the name of the operation suggests. N-ary links must be handled by giving the user a choice of components to navigate to. This is particularly interesting with regard to reverse navigation. It is feasible to navigate from one node to another across a link, then to reverse navigate but to end up at a different component. However, this is consistent with the semantics of the n-ary links. N-ary links establish a common association between two sets of components, not just in the type of association, but in the instance of the association as well.

Definition 14. Given a perspective $P(\pi, S)$, let $c \in N(\pi)$ be a node (either a component or link class). A *browsing* operation $\mathcal{B}$ returns all of the relative object instances for c within the HOMIS that are included in one of the pattern instances $s \in S$:

$$\mathcal{B}_c(P) = \{o | o = \mathcal{A}(c) \wedge o \in N(s)\}$$

Definition 15. Given a perspective $P(\pi, S)$, let o be a displayed object (either component or link) in the instance $s \in S$, and let a be one of its complex attributes. Then a *navigation* operation $\mathcal{N}$ returns the linked objects:

$$\mathcal{N}_{a(o)}(P) = \{o' | \exists Y, Z \text{ such that } o \in Y \wedge o' \in Z \wedge (Y, a, Z) \in \mathcal{L} \wedge o' \in N(s)\}$$

Definition 16. Given a perspective $P(\pi, S)$, let o be a displayed object (either component or link) in the instance $s \in S$, and let a be one of its complex attributes. Then a *reverse navigation* operation $\mathcal{N}^{-1}$ returns the linked objects:

$\mathcal{N}^{-1}_{a(o)}(P) = \{o' \mid \exists Y, Z \text{ such that } o \in Y \wedge o' \in Z \wedge (Z, a, Y) \in \mathcal{L} \wedge o' \in N(s)\}$

The navigation and reverse navigation definitions differ substantially from those in the original model in that that the relationships in $\mathcal{L}$ are of the form (set, association, set) instead of (element, association, element).

D. USER INTERFACE DESIGN ISSUES

1. Consistency

Consistency of the user interface is vital to reducing the cognitive load on the user. Since querying will primarily be done through successive filters, it is important the coding of the symbols shown the user remains the same throughout each iteration. Colors and shapes represent important information to the user as to what type of object is being shown [Galitz, 1993]. Therefore, if rectangles are used for object instances in one version of the graph they must be used in all successive versions. Likewise color changes must not occur. It could be possible to create run-time applications that assign different color codes to each level of abstraction in a class hierarchy. As the user moves up and down these levels in setting perspectives and filters all interactions must retain the same color coding. Therefore, color coding should either be added to the model in the storage layer or should be calculated consistently among run-time applications from storage layer attributes.

2. Coding

In the examples produced so far, only simple coding is used. The model could be enhanced to code different classes with different shapes or colors. An example might be to include an icon with each class type. In this way people could be represented by images of people, text can be indicated by some sort of text icon. Different types of links could be distinguished by different colors. Inheritance is already coded in the schema graph

through a thicker line than that used for other edges. Therefore, this technique should not also be used for other meanings. [Mayhew, 1992]

One coding aspect that is commonly used in hypermedia research is length of edge in a graph to indicate the closeness in terms of association of two pieces of information. Longer edges imply that the two nodes are not as closely related as shorter edges. Given the difficulty in finding a graph display that will draw nodes and edges in a visible manner given that different analysts are adding information under different perspectives, this does not appear useful for dynamic hypermedia. One coding option for these "fish-eye views" [Tochtermann. and Dittrich, 1992] is to use color intensity. The more intense the component, link, and edges, the more closely related the information.

Particular implementation of HOPE architectures are going to need to establish coding conventions for developers of run-time software. The integration of already produced software could complicate this issue, but these modules are more likely to be dealing with content rather than the hypermedia structures themselves.

3. Access to Schema

Many types of users might use the hypermedia systems built using the HOPE architecture. Not all may be comfortable with the graph view. The run-time layer offers the opportunity for a wide variety of applications to be produced that interact with individual user types in a manner that is appropriate for their job. Those who will only pose criticisms can interact through a simple form window and never see the hypermedia structures underneath. Those who are responsible for connecting items could view individual components and have link options made available through a menu. Once again they do not need to view the hypergraph. However there is considerable research that indicates that providing the hypergraph perspectives to the user will actually make their job easier [Nielsen, 1995].

VI. APPLICATION TO SOFTWARE REQUIREMENTS ANALYSIS

A. COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

Software engineering is an analytical activity that fits the pattern described earlier. In particular the requirements analysis process, where one must trace user needs to software requirements and requirements to design decisions and components is a natural fit for this pattern. A model for such analysis has been developed for use with the Computer Aided Prototyping System (CAPS) [Ibrahim, 1996]. CAPS is an integration of software engineering tools developed to support research in software engineering and to assist software engineers in the development of real-time prototypes. The software lifecycle supported is shown in the following figure.

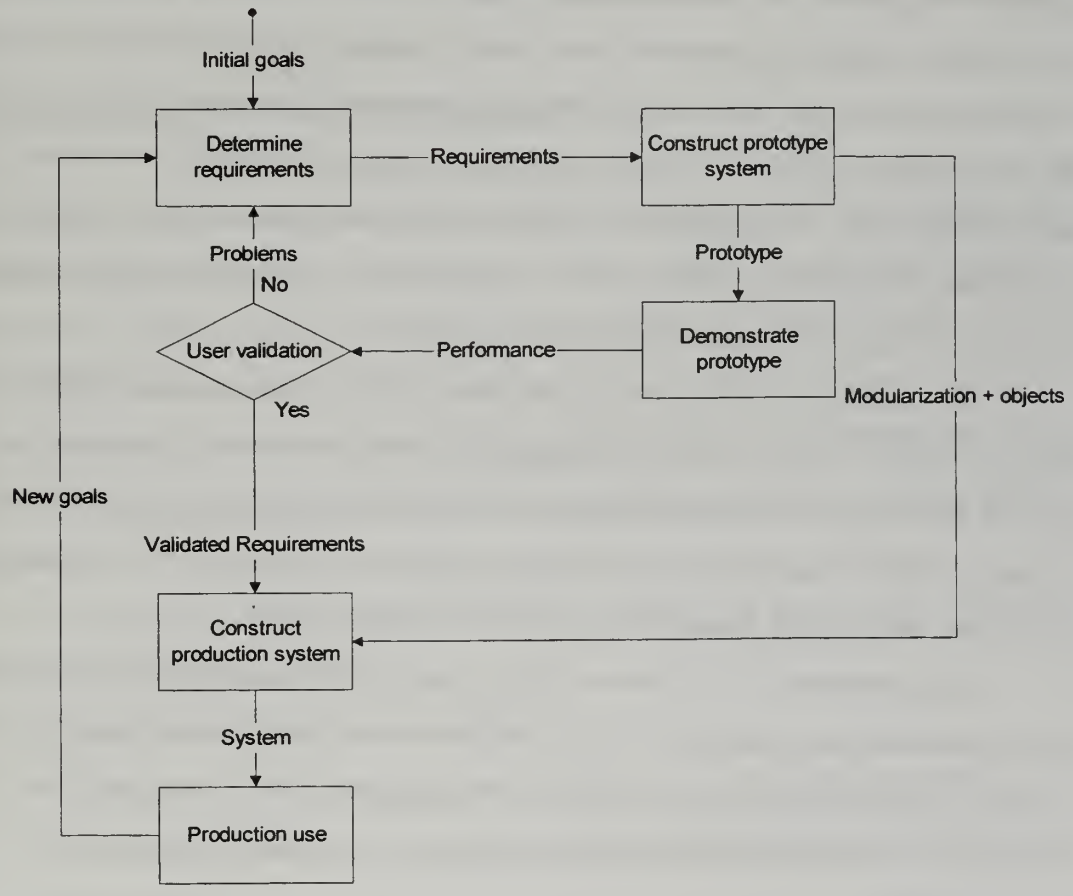

Figure 6.1 Prototyping life cycle. [Luqi and Berzins, 1988]

In this model, the requirements are not assumed correct and complete at the beginning of the project. Instead, an iterative process of developing prototypes, demonstrating them to the users, and evaluating their responses is used to incrementally improve the requirements set. An evolution process is needed to manage the analysis of these inputs and ensure that improvement is made with each cycle. Formal model-based tools to support such a process are vital to keeping the integrity of a project through each cycle. This is particularly important in large projects where multiple analysts and designers work concurrently. [Luqi and Goguen, 1997]

The hypergraph [Luqi, et. al., 1994] model of software evolution was introduced in an earlier form as a graph model [Luqi, 1990]. In both models, there are two main types of elements that are the primitives from which all others are described. These are software components and evolution steps. Initially, only components and steps that dealt with software design were described in the model. Ibrahim extended the model to include components and steps relative to the requirements analysis aspects of the prototyping lifecycle [Ibrahim, 1996]. Ibrahim's component extensions included the actors in the lifecycle processes. In this thesis the model is described in terms of an object-oriented hypermedia architecture [Schwabe, et. al., 1995]. Steps are further described in terms of run-time layer applications that support analysis efforts.

The Prototype System Design Language (PSDL) [Luqi, et. al., 1988] forms the basis for CAPS. PSDL models the timing and control constraints of real-time software and can be used to automatically generate Ada code for prototype implementations of a system. CAPS is structured as an integrated collection of software tools grouped in the subsystems shown in the figure below [Luqi and Ketabchi, 1988].

The Evolution Control System (ECS) controls and manages software components and development team interactions. The initial version of the ECS handled version control of PSDL components and the scheduling of developer tasks. [Badr and Luqi, 1994] Subsequent work added requirements management steps and artifacts to the system [Ibrahim, 1996].

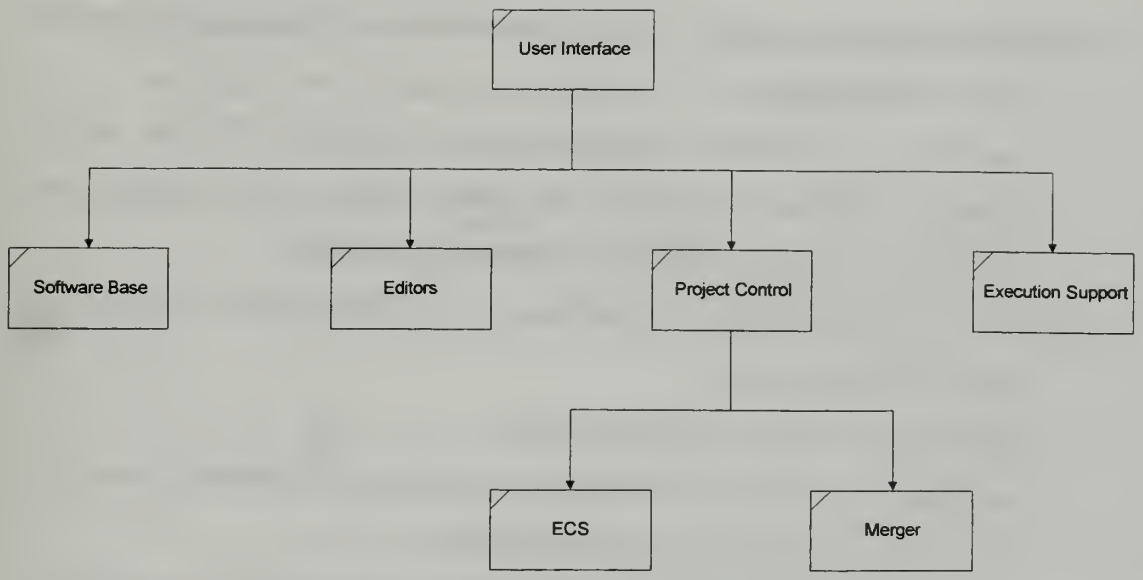

Figure 6.2 CAPS Structure [Ibrahim, 1996]

B. DECISION SUPPORT MECHANISMS FOR CAPS

Software evolution encompasses the activities that change a software system and the relationships among those activities. These include requirements analysis, modifications to existing components, and many other activities [Luqi and Goguen, 1997]. Change occurs throughout the lifecycle of software products. Software evolution processes manage and steer such change. In the model used by CAPS, change motivates the use of a prototyping process that interleaves evolution activities with development [Luqi, 1989].

The evolution control system of CAPS is intended to support the following capabilities listed in [Ibrahim, 1996]:

1. Planning the prototype demonstration.
2. Mapping user criticisms into the primitives of a formalized model to be analyzed and elaborated so that they can be synthesized into a set of issues to be resolved.
3. Analyzing alternatives available and choosing among them to make necessary modifications in the design to resolve the open issues.

4. Creating analysis activities. Plan and execute these activities when the needed resources are available.
5. Controlling the evolution of the requirement components that are directly affected, as well as propagating the implied effects of the changes and configuring the whole requirements hierarchy accordingly.
6. Propagating any changes in requirements to the affected parts of the system design and implementation.
7. Coordinating the effort of the design team.
8. Controlling versioning and configuration management to faithfully reflect the intended effect of the dynamic ongoing changes.

Actors and use cases developed to better understand these requirements are provided in Appendices A and B respectively. The methods and formats used are derived from [Jacobsen, et. al., 1992], [Rumbaugh, et. al., 1991], and [Awad, et. al., 1996].

Steps are initiated automatically through dependencies in the model. As will be shown, the ability of the HOPE architecture to support links-to-links is used to link steps to the associations that cause their initiation.

All steps have states. The states used are listed below and shown in the diagram that follows [Badr and Luqi, 1994]:

1. Proposed: The initial state of a newly created step. In this state a step is subjected to cost and benefit analysis.
2. Approved: The work to be accomplished has been approved by the manager and scheduling attributes such as priorities, required skills, and effort estimates have been determined.
3. Scheduled: The step has been scheduled for implementation and expected starting and finish times are calculated.
4. Assigned: A step has been assigned to a designer or analyst and the work is in progress.
5. Completed: In this state the output of the step has been verified, and an immutable version has been entered into the project database.

6. Abandoned: The step has been cancelled before it has been completed. It is reachable from all other states except the "Completed" state.

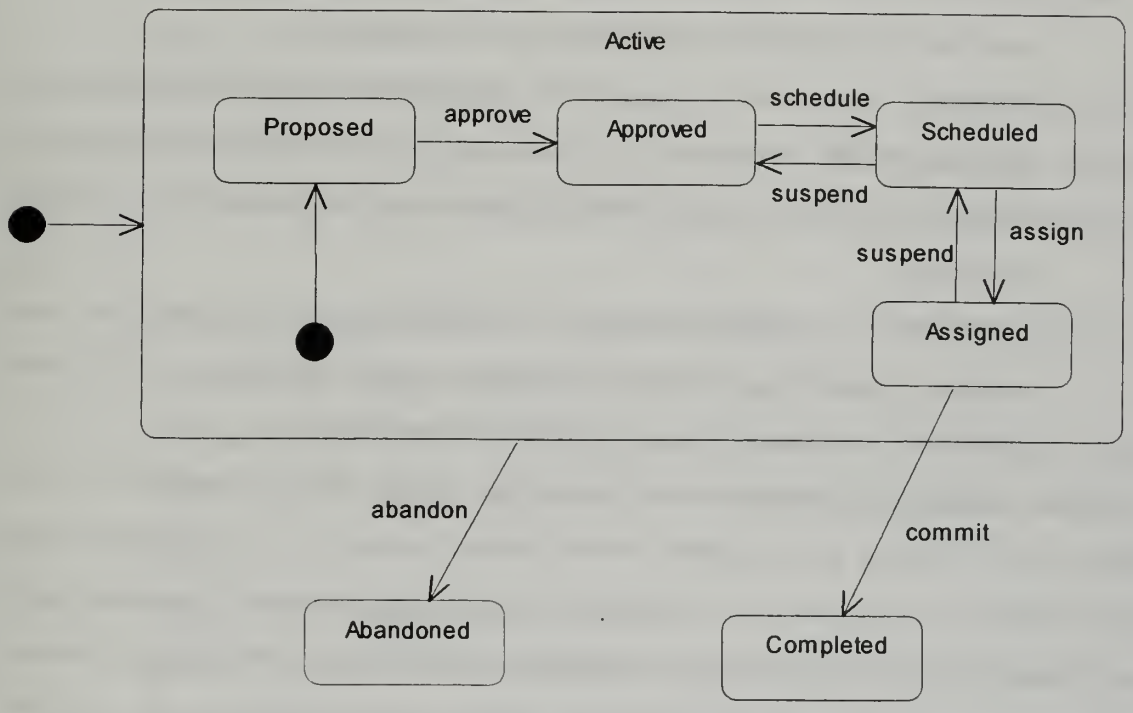

Figure 6.3 ECS Step States

The inputs and outputs of these steps are described below [Ibrahim, 1996].

1. Criticisms: These are comments provided by users concerning their evaluation of a prototype resulting from a demonstration. The criticism can be linked by the user to a particular scenario, demonstration, or prototype.
2. Issues: Issues are created using criticisms posed by users. They are one of the intermediate results of analyzing criticisms. Issues represent the problems to be solved.
3. Requirements: These describe alternative means by which issues can be solved. An alternative is selected when a particular set of requirements is *approved* by the project manager.
4. Module: Modules represent the design elements of the system. These can be PSDL components, Ada code, or any other form of implementation of a set of requirements.

5. Demonstration: A demonstration is set up by a project manager. It incorporates a series of test scenarios, and is intended to demonstrate a particular set of requirements that have been implemented.
6. Scenario: A scenario is a series of user-system interactions designed to demonstrate the satisfaction of a particular set of requirements solving a particular set of issues. It is intended to guide a user's examination of a system prototype.
7. Prototype: A prototype represents a collection of design modules that together represent a version of the system being developed. The entire prototype is examined by the user during a demonstration.

All of these objects are immutable once the step that creates them is completed. Once created their content is not permitted to be changed in any way. Their status may however change. It is for this reason that status is captured as a separate component node class. This provides a history of a project through its artifacts. In order to effect change, new versions are created. Links between artifacts are permitted to be changed. Associations among the elements are the primary contributions of the analysts involved. One of the challenges in using hypermedia for software engineering is determining how to handle changes to associations when there are multiple versions of a product. The choice made here has been to replicate links to each new version, then only change the links on the current versions being analyzed. Links to previous versions become immutable when the steps producing those previous versions enter the *completed* state.

The associations used in the model are:

1. PartOf: This type of link connects objects of the same type and represents the decomposition structure of software components or steps.
2. Uses: Links components to show that either the semantics or implementation of one is affected by the other.
3. PrimaryInput: Links an object to be updated to a step that will create a new version of the object.
4. SecondaryInput: Links an object to steps that need read-only access to the object.

5. Affects: Links a criticism to an issue or an issue to a requirements component that it affects. This is relation is explicitly declared by an analyst or designer.
6. Poses: Links a user to a criticism that he or she poses.

These associations are supplemented with some new ones that better express the relationships between items in a hypermedia model. Some of these relationships did not make sense in a non-hypermedia interpretation as originally created. These associations are:

1. Criticizes: Indicates the object of the user's criticism. Different from affects in that it does not cause a ripple effect when a change occurs.
2. AssignedTo: This was a treated as an attribute within an object in the object model and was left out as an inter-object relationship between steps and people.
3. Collects: An inverse of secondary input.
4. Spawns: Unique to this hypermedia approach to immutable objects and therefore not in Ibrahim's or Badr's models. Spawns is a relationship between a version of an object and a future version that is created from it. By using this relationship, links to the parent version can be used to navigate to newer versions. Likewise, algorithms can decide when to copy links possessed by the parent object to the spawned object.
5. Initiates: Connects a step to the association that caused it to be necessary. All previous models described the steps as associations. here they are components and initiates is the association to a new step (implemented as link-to-link). For example, when a user submits a criticism of a prototype. an analysis step finding the issues related to the criticism must be initiated The analysis step is a component that has been initiated by the action taken by the user to submit the criticism. Therefore, it is related to the association between the user and the criticism and thus a link to a link (see Figure 6.10)
6. hasState: Using the state pattern [Gamma, et. al., 1995] [Pree, 1995], state is represented as an object separate from the step that it is describing. The association hasState provides the connection.

7. Demonstrates: This is a special case of *uses*.

8. Tests: Like *demonstrates* this is a special case of *uses*.

The schema that creates these objects and relationships is shown below in OMT notation.

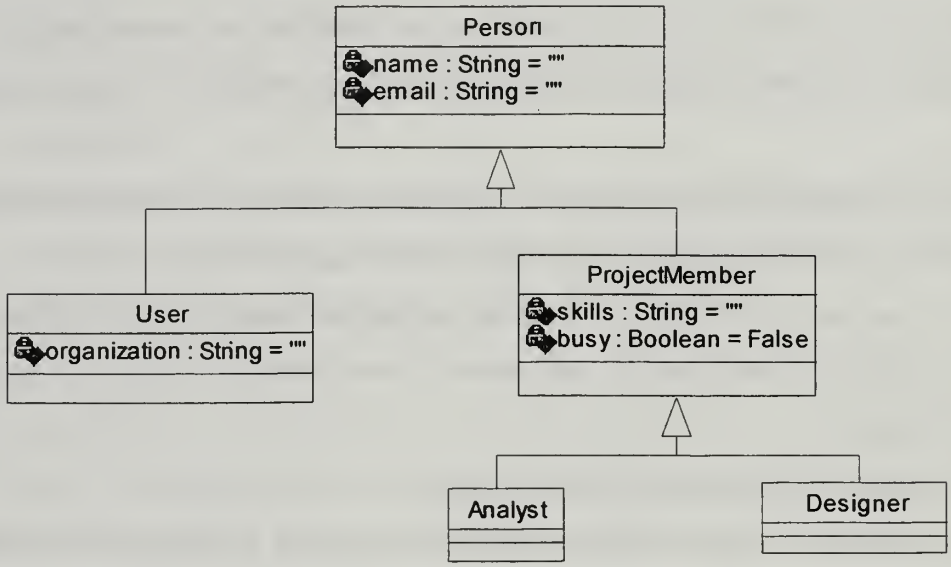

Figure 6.4 Humans

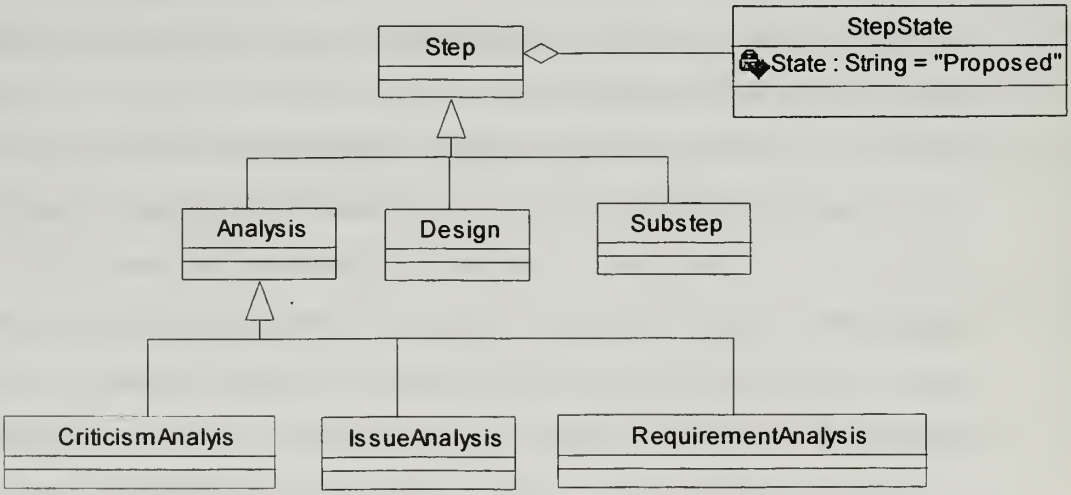

Figure 6.5 Steps

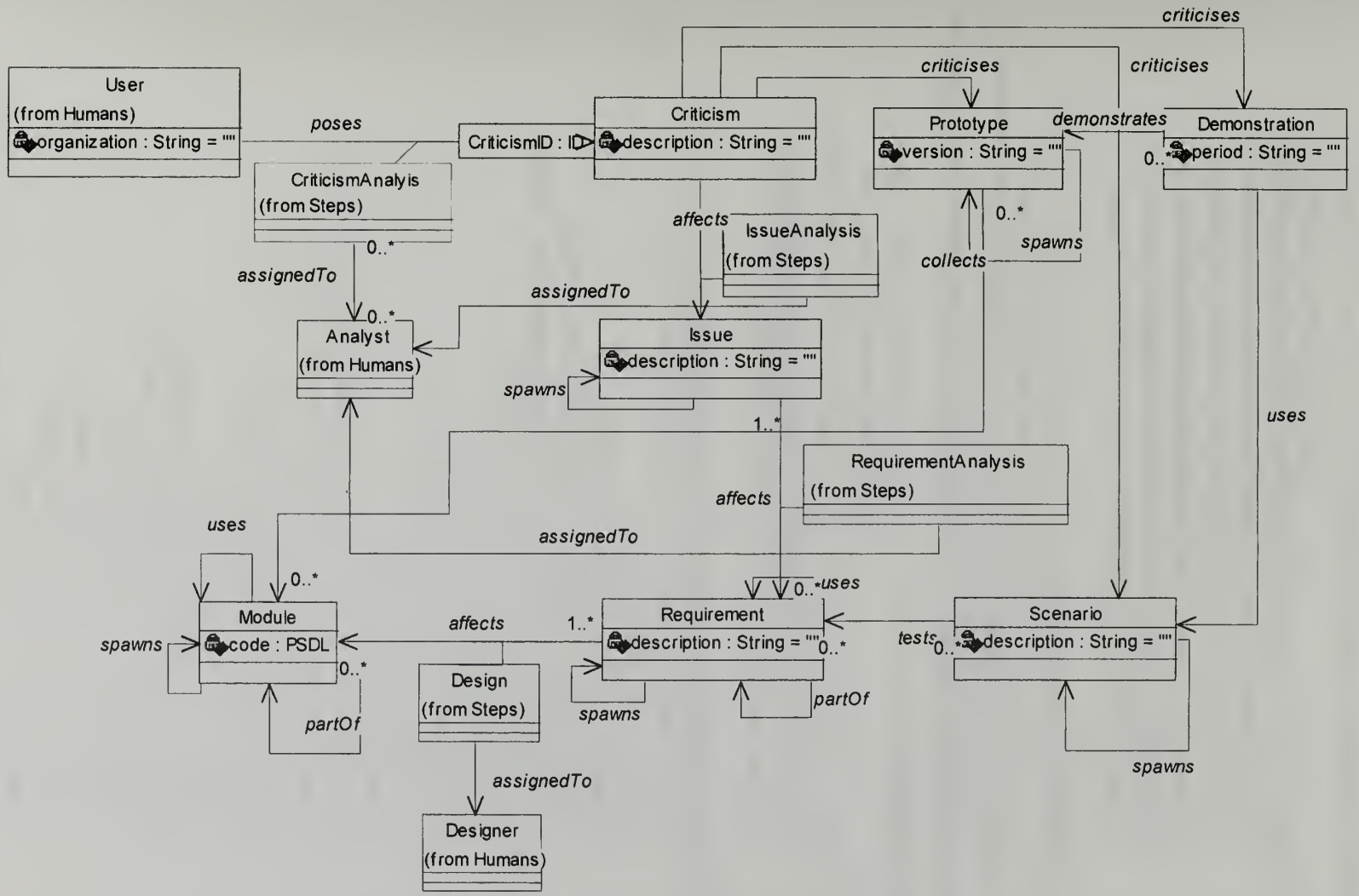

Figure 6.6 Artifacts

As stated in Chapter III, the conceptual schema for this hyperobject multimedia analysis system is $\Sigma = (C, T, A, \mathscr{P}, \mathscr{H}, \mathscr{A})$.

C = {Person, User, Analyst, Designer, Step, Analysis, Design, Substep, CriticismAnalysis, IssueAnalysis, RequirementAnalysis, poses, Criticism, criticizes, assignedTo, affects, collects, uses, Issue, Prototype, Demonstration, spawns, demonstrates, Module, Requirement, partOf, Scenario, initiates, StepState, ProjectMember, tests, hasState}

T = {String, PSDL, Boolean}

Note that *T* is being kept minimal here. No further types are required to implement the prototyping method, though they may be used to enhance the information presented to the analysts.

A = {poses, criticizes, assignedTo, affects, collects, uses, spawns, partOf, name, email, description, period, version, code, organization, initiates, hasState, state, skills, busy, demonstrates, tests}

$\mathscr{P}$ = {(User, organization, String), (Person, name, String), (Person, email, String), (Criticism, description, String), ({User}, poses, {Criticism}), ({poses}, initiates, {CriticismAnalysis}), ({Step}, hasState, {StepState}), (StepState, state, String), (ProjectMember, skills, String), (ProjectMember, busy, Boolean), ({CriticismAnalysis}, assignedTo, {Analyst}), ({Criticism}, criticizes, {Prototype}), ({Criticism}, criticizes, {Demonstration}), ({Criticism}, criticizes, {Scenario}), (Prototype, version, String), (Demonstration, period, String), (Scenario, description, String), (Issue, description, String), ({Issue}, spawns, {Issue}) ({affects}, initiates, {IssueAnalysis}), ({IssueAnalysis}, assignedTo, {Analyst}), ({Issue}, affects, {Requirement}), ({affects}, initiates, {RequirementAnalysis}) ({RequirementAnalysis}, assignedTo, {Analyst}) ({Requirement}, uses, {Requirement}) ({Requirement}, partOf, {Requirement}), ({Requirement}, spawns, {Requirement}), (Requirement, description, String), ({Requirement}, affects, {Module}), (Module, code, PSDL),

({affects}, initiates, {Design}),
({Design}, assignedTo, {Designer}),
({Module}, uses, {Module}), ({Module}, spawns, {Module}),
({Module}, partOf, {Module}),
({Demonstration}, demonstrates, {Prototype}),
({Prototype}, spawns, {Prototype}),
({Prototype}, collects, {Module}),
({Demonstration}, uses, {Scenario})
({Scenario}, spawns, {Scenario}),
({Scenario}, tests, {Requirement}), ({Step}, uses, {Substep})}

$\mathcal{H}$ = *{(User, Person), (CriticismAnalysis, Analysis), (Analysis, Step),*
(Analyst, ProjectMember), (ProjectMember, Person),
(IssueAnalysis, Analysis), (RequirementAnalysis, Analysis),
(Design, Step), (Designer, ProjectMember), (Substep, Step)}

$\mathcal{A}$ = *{poses, initiates, hasState, assignedTo, criticizes, affects, spawns,*
uses, partOf, demonstrates, tests}

1. Requirements Analysis Process Overview

The analysis phase of the prototype cycle is shown in the figure below [Ibrahim, 1996].

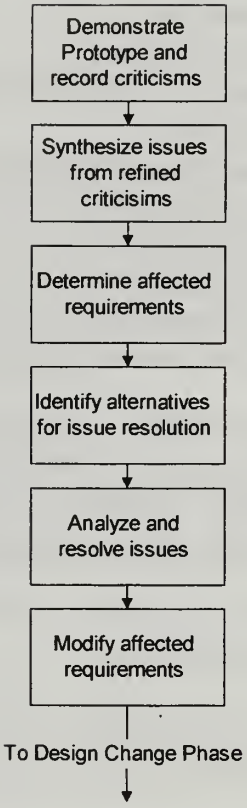

Figure 6.7 Requirements Analysis Process Schematic

2. Initial Requirements

The lifecycle of a project starts when an initial set of requirements has been collected. For the sake of illustration, we will create a HOMIS to be the project database of a prototyping project. There is one analyst, one designer, and two users for this prototyping project. An initial set of two requirements has been collected and entered into the system. The HOMIS for this project would start off as described below.

$M = (\Sigma, O, \mathcal{I}, \mathcal{L})$

Σ is described above.

$O = \{user1, user2, anal1, designer1, req1, req2\}$

$\mathcal{I}$= {(user1, User), (user2, User), (anal1, Analyst), (designer1, Designer), (req1, Requirement), (req2, Requirement)}

$\mathcal{L}$= {(user1, name, string1), (user1, email,string2), (user1, organization, string3), (user2, name, string4), (user2, email, string5), (user2, organization, string6), (anal1, name, string7), (anal1, email, string8), (anal1, skills, string9), (anal1, busy, boolean1), (designer1, name, string10), (designer1, email, string11), (designer1, skills, string12), (designer1, busy, boolean2), (req1, description, string13), (req2, description, string14) }

typen (e.g., *string14*) indicates an anchor to an instance of a particular type *t*. This ties an object to a particular *V(t)* found in the *within-content* layer of HOPE.

The first steps to be performed are design steps. If one design step is being performed to handle both of the initial requirements, then after the assignment to the designer the sets would look like the following (relevant to their initial states).

$O = O \cup$ {design1, stepstate1, hasState1, assignment1}

$\mathcal{I}$= $\mathcal{I} \cup$ {(design1, Design), (stepstate1, StepState), (hasState1, hasState), (assignment1, assignedTo)}

$\mathcal{L}$= $\mathcal{L} \cup$ {({design1}, assignment1, {designer1}), ({design1, hasState1, {stepstate1}), (stepstate1, state, string15)}

The anchor *stepstate1* would point to a state object that represents that the step has been assigned.

After the design of the modules has been completed, the check in of the modules changes the links and content of the components. The state of the design step is changed to *completed*, but no change occurs in the sets above for that to happen. *stepstate1* is a mutable object in the *within content* layer and can be modified once retrieved. The final action taken is that the module that is entered into the project database is connected to the requirements that affected it. The assignment connection is left as it was as a record of who performed the design. Since the assignment is from a step with a completed state now, we can easily filter the display to prevent its appearance if desired.

$O = O \cup$ {module1, affects1, affects2, initiates1}

$\mathcal{I} = \mathcal{I} \cup \{$*(module1, Module), (affects1, affects), (affects1, affects), (initiates1, initiates)*$\}$

$\mathcal{L} = \mathcal{L} \cup \{$*({req1}, affects1, {module1}), ({req2}, affects2, {module1}), (module1, code, PSDL1), ({affects1, affects2}, initiates1, {design1})*$\}$

At the end of this step the hypergraph representing these sets would appear as follows:

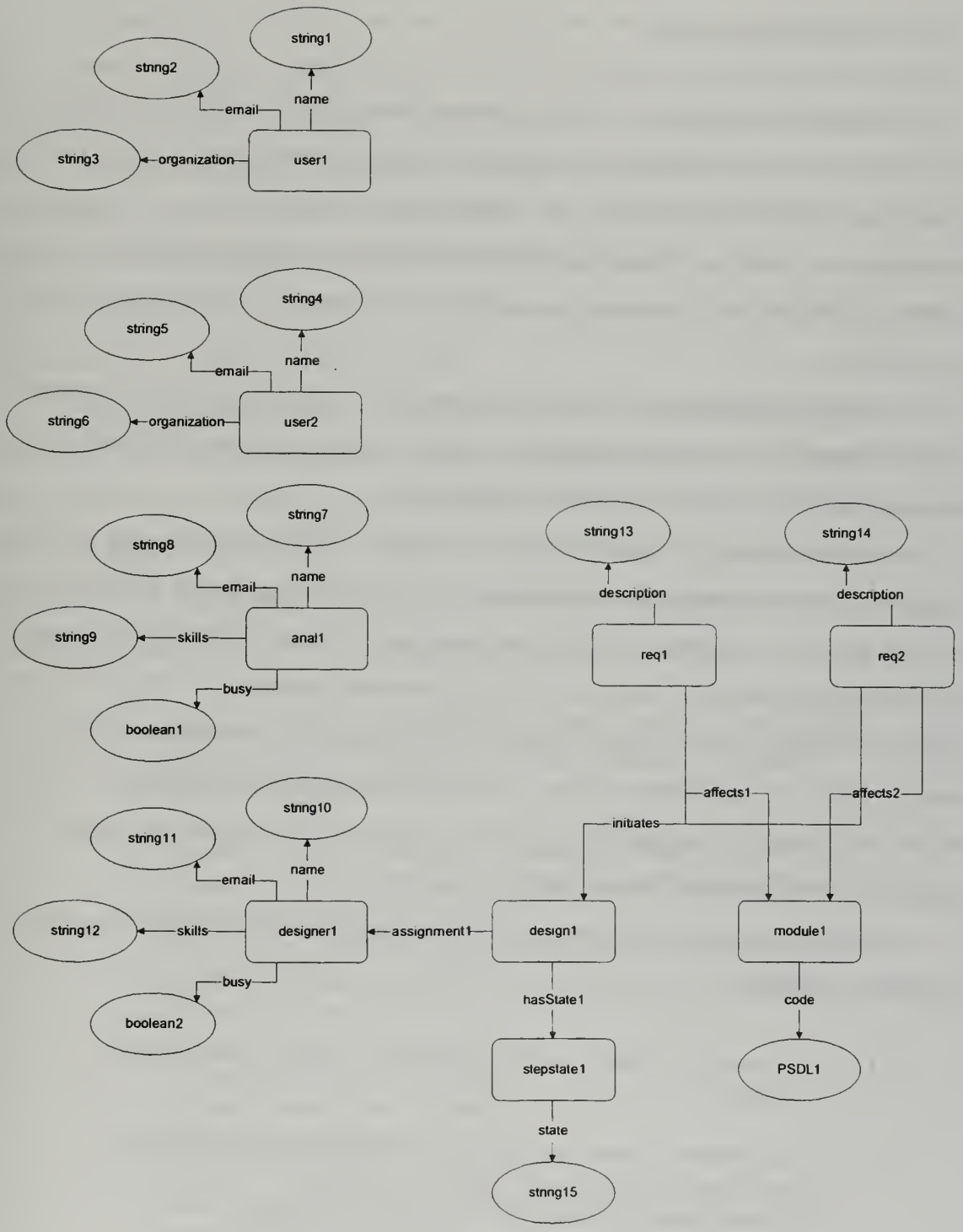

Figure 6.8 Hypergraph after initial development.

3. Demonstration Step

As specified in [Ibrahim, 1996] the demonstration step is to proceed as follows. Test scenarios are run with the users using the prototype to be reviewed. Their criticisms are recorded, reviewed for accuracy, and recorded in the project database. These are to be associated with the users who posed them along with any amplifying information. The comments must also be linked to the scenarios and version of the prototype being demonstrated.

Specific use cases are provided in Appendix B. One aspect that immediately becomes apparent in use case analysis of this step is how much easier preparing for the demonstration becomes with hypermedia tools available.

Demonstrations for users should be matched to their concerns and to their expertise. In a non-hypermedia environment, one would need to search for issues of a particular nature, or criticisms posed by a particular user. Queries would then be issued to find requirements traced to those, or involving the same keywords. In the hypermedia architecture being proposed here, one could merely link a demonstration object to the issues. Agents that can follow particular types of links and recognize particular structures can find the scenarios that have been previously linked to the requirements found to be associated with the issues of interest to a particular user.

Before the demonstration can proceed, test scenarios need to be developed and a demonstration planned.

$$\mathcal{O} = \mathcal{O} \cup \{scenario1, demo1, tests1, tests2, uses1, proto1, demonstrates1, collects1\}$$

$$\mathcal{I} = \mathcal{I} \cup \{(scenario1, Scenario), (demo1, Demonstration), (tests1, tests), (tests2, tests), (uses1, uses), (proto1, Prototype), (demonstrates1, demonstrates), (collects1, collects)\}$$

$$\mathcal{L} = \mathcal{L} \cup \{(\{scenario1\}, tests1, \{req1\}), (\{scenario1\}, tests2, \{req2\}), (scenario1, description, string16), (demo1, period, string17), (\{demo1\}, uses1, \{scenario1\}), (\{demo1\}, demonstrates1, \{proto1\}), (proto1, version, string18), (\{proto1\}, collects, \{module1\})\}$$

The semantics of when to use an n-ary link and when to use separate links deal with whether or not the *n* relationships will ever be divided into individual relationships. In the initial development step, the relationship of the two requirements to the single module initiated the single design step. Therefore, a single link that included both requirements on the *from* end was appropriate. In developing the test scenario and planning the demonstration, the desire to reuse the scenario in the future and the recognition that either one or both of the first two requirements could change (in future versions) leads to the use of individual links.

Once the demonstration has been executed the user enters criticisms that are linked by run-time tools to the appropriate scenarios, demonstration, or prototype, depending how specific the user wishes to be. The creation of criticisms *initiates* criticism analysis steps that begin in the *proposed* state. After reviewing the criticisms for clarity, plausibility, and consistency, the steps are moved into the *approved* state, then get *scheduled*. Finally, the analysis of criticisms is *assignedTo* an *Analyst*.

$O = O \cup$ {*poses1, criticism1, criticizes1, initiates2, critanal1, hasState2, stepstate2, poses2, criticism2, critisizes2, initiates3, critanal2, hasState3, stepstate3*}

$\mathcal{I} = \mathcal{I} \cup$ {*(poses1, poses), (criticism1, Criticism), (critisizes1, criticizes), (initiates2, initiates), (critanal1, CriticismAnalysis), (hasState2, hasState), (stepstate2, StepState), (poses2, poses), (criticism2, Criticism), (critisizes2, criticizes), (initiates3, initiates), (critanal2, CriticismAnalysis), (hasState3, hasState), (stepstate3, StepState)*}

$\mathcal{L} = \mathcal{L} \cup$ {*({user1}, poses1, {criticism1}), (criticism1, description, string19), ({criticism1}, criticizes, {proto1}), ({criticizes1}, initiates2, {critanal1}), ({critanal1}, hasState2, {stepstate2}), (stepstate2, state, string21), ({user2}, poses2, {criticism2}), (criticism2, description, string20), ({criticism2}, criticizes, {scenario1}), ({criticizes2}, initiates3, {critanal2}), ({critanal2}, hasState3, {stepstate3}), (stepstate3, state, string22), ({critanal2}, assignment2, {anal1})*}

The hypergraph could appear as in the figure below.

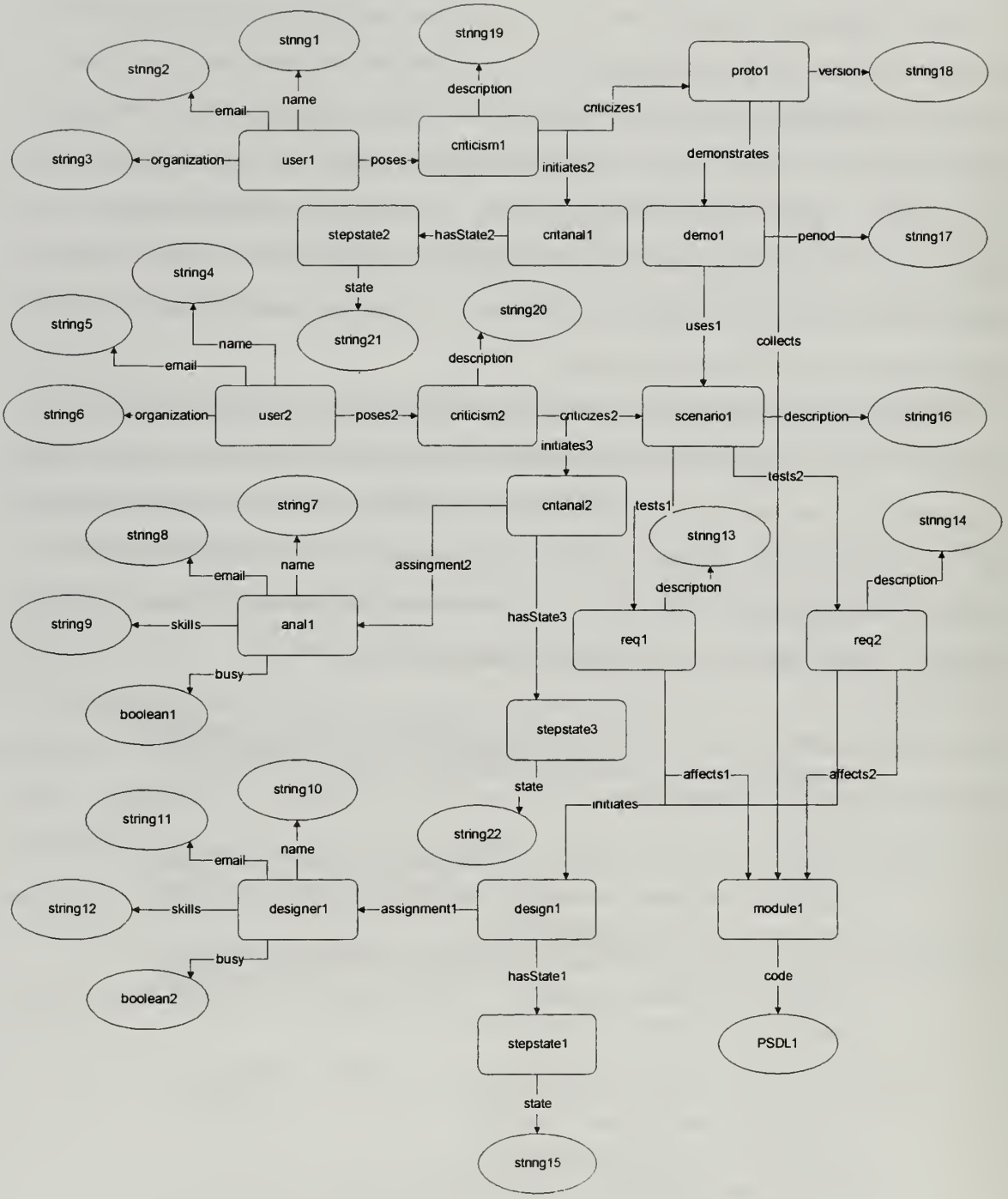

Figure 6.9 Hypergraph after the demonstration step

Overview diagrams like the one above are too complex for the user to make use of. As the number of components and links increases it becomes difficult for the analyst to utilize such diagrams. For this reason, diagrams that follow will use perspectives to limit

the hypergraph to the information needed. This will help simplify the diagram and help focus attention on the most important points. Likewise, analysts would do the same.

4. Criticism Analysis Step

When the demonstration step is completed, and criticisms have been assigned, analysis is performed to extract issues from the criticisms. If issues already exist, the analyst wants to connect criticisms to those issues if possible. Therefore, issues and criticisms are both part of the perspective of the analyst. The user's themselves are part of the perspective as it is possible that criticisms from multiple users contradict each other or contradict issues brought about by past criticisms of other users. Using HOPE tools, we can predefine a perspective for criticism analysis. We do this using the schema and "clicking" on those component and link classes that are of interest to the analyst in this situation. Those selected are:

- poses
- criticizes
- Issue
- Criticism

If we had more entries in the project database at this point a filter would be necessary as well. We will have entries to filter out later in the scenario. The resulting hypergraph is shown below.

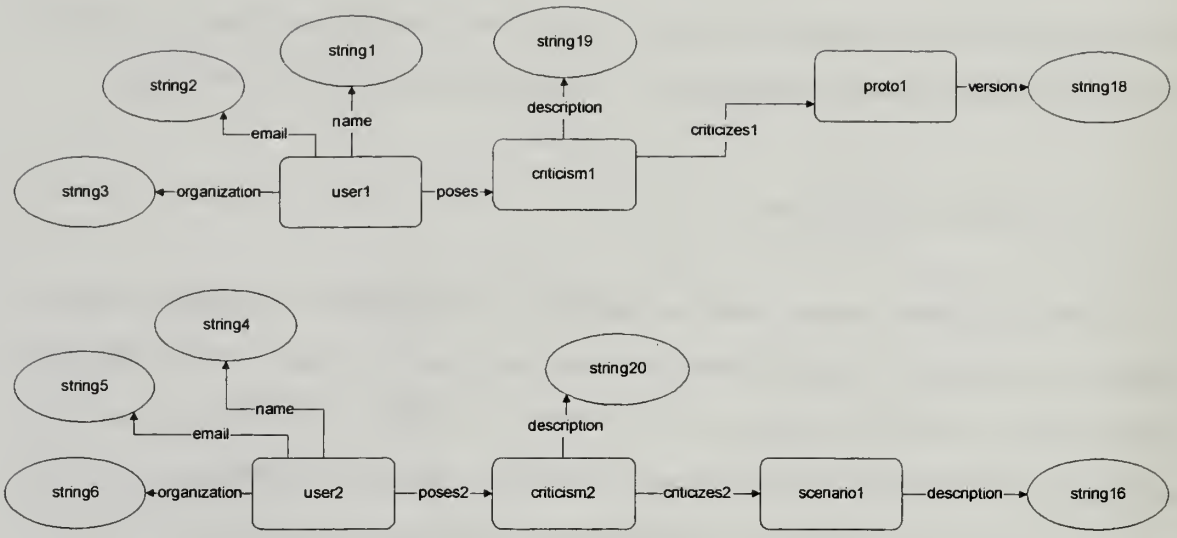

Figure 6.10 Criticism Analysis Perspective

The analyst finds the issue in the criticism assigned and creates an issue object. The criticism is linked to the issue using the *affects* relationship and a new IssueAnalysis is initiated based on that action. The IssueAnalysis begins in the proposed state and is eventually scheduled and assigned. The criticism posed by *user1* is then assigned to the analyst who decides that the criticism affects the same issue and does not require the issue to be reworked. The criticism is joined to the same link used by the other criticism in order to prevent a separate issue analysis from being created. The alternative if the two were different but revolved around the same issue would be to spawn a new issue and attach the second criticism to that new version of the issue. The run-time application then knows to attach links for those relationships of the previous version to the new version. The criticism analysis is now completed.

$$O = O \cup \{affects3,\ issue1,\ initiates4,\ issueanal1,\ hasState4,\ stepstate4\}$$

$$\mathcal{I} = \mathcal{I} \cup \{(affects3,\ affects),\ (issue1,\ Issue),\ (initiates4,\ initiates),\ (issueanal1,\ IssueAnalysis),\ (hasState4,\ hasState),\ (stepstate4,\ StepState)\}$$

$$\mathcal{L} = \mathcal{L} \cup \{(\{criticism2,\ criticism1\},\ affects3,\ \{issue1\}),\ (\{affects3\},\ initiates4,\ \{issueanal1\}),\ (\{issueanal4\},\ hasState4,\ \{stepstate4\}),$$

(stepstate4, state, string23), ({critanal2}, assignment3, {anal1}), ({issueanal1}, assignment4, {anal1})}

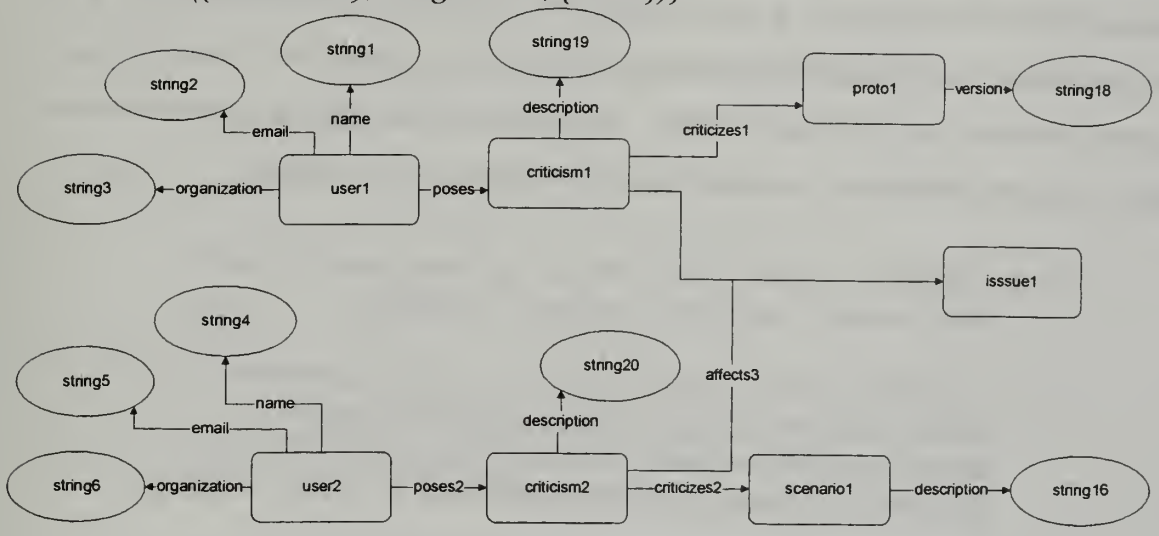

Figure 6.11 After Criticism Analysis

5. Issue Analysis Step

A new perspective is needed for this step. Issues and requirements are what are important here. There is less of a connection to the users, but the criticisms posed and the connection their relationship to the prototype and scenarios may still be useful to the analyst. The perspective for issue analysis is shown below. The perspective pattern classes are:

$N(\pi)$ = *{Criticism, Prototype, Scenario, demonstrates, Issue, Module, Requirement, tests, collects, criticizes}*

The purpose of this step is to determine what requirements are affected by the issues presented. There are multiple ways to solve any issue, so many alternatives may need to be explored by the analyst. Here the benefit of public vs. private links is shown. By keeping links private, an analyst can explore multiple solutions without affecting other workers. Decision aids such as those described in the following section can be used to choose the best option. Once an option is chosen, those links are made public. Again the value added by the analyst is mostly in the creation of links.

Here the issue is determined to be best solved by modifying requirement1. Requirements are immutable so a new version of the requirement must be spawned. Run-time applications performing this function must know what connections to keep constant and what to move over to the new version. The analyst decides that the scenario does not fully test the new requirement, so a new scenario will have to be developed.

$O = O \cup \{affects4, spawns1, req3, affects5, affects6, spawns2, tests3, tests4, reqanal1, hasState5, stepstate5, initiates5\}$

$\mathcal{I} = \mathcal{I} \cup \{(affects4, affects), (spawns1, spawns), (req3, Requirement), (affects5, affects), (affects6, affects), (spawns2, spawns), (tests3, tests), (tests4, tests), (reqanal1, RequirementAnalysis), (hasState5, hasState), (stepstate5, StepState), (initiates5, initiates)\}$

$\mathcal{L} = \mathcal{L} \cup \{(\{req1\}, spawns1, \{req3\}), (\{issue1\}, affects4, \{req3\}), (req3, description, string24), (\{req3\}, affects5, \{module1\}), (\{scenario1\}, spawns2, \{scenario2\}), (scenario2, description, string25), (\{scenario2\}, tests3, \{req3\}), (\{scenario2\}, tests4, \{req2\}), (\{affects5\}, initiates5, \{reqanal1\}), (\{reqanal1\}, hasState5, \{stepstate5\}), (stepstate5, state, string26)\}$

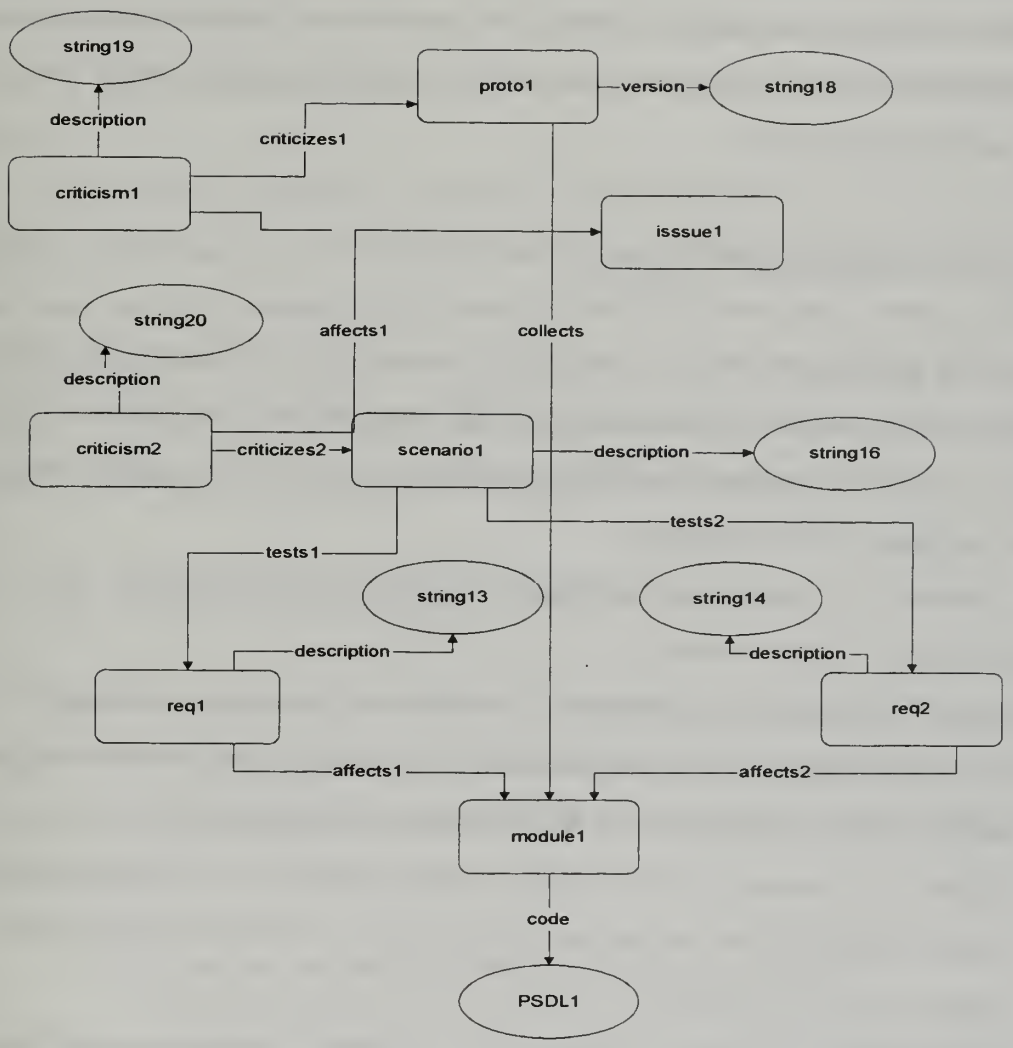

Figure 6.12 Issue Analysis Perspective

C. DECISION SUPPORT TOOLS USING HYPERMEDIA APPROACHES

The remaining steps of analysis and design proceed in a similar fashion to those described above. Unique perspective patterns may be employed for each step. These are stored and retrieved allowing the analyst to easily work on the job at hand. This section describes some of the perspectives necessary to assist stakeholders of a prototype system being developed. One of the benefits of perspectives is that it may be possible to handle

many of the anticipated user interactions through a perspective view, eliminating the need to write code for additional run-time tools. There will be situations where it is desirable to have a very job-specific run-time application available to guide the user through a set of interactions with the hypermedia.

1. User Examination of Criticisms

Users will periodically want to check on the status of the criticisms they have posed. They will want to know if the criticism has been analyzed, what the issue was that was created or modified based on the criticism, and what version of the system might demonstrate correction of their concerns.

By selecting the following components and links from perspective view a perspective for the user can be created:

- Poses. This causes user, criticism and criticism analysis nodes to be brought in, but does not bring in links between criticisms and scenarios, demonstrations and prototypes. The state nodes for the analysis are also selected.
- Affects. By selecting the affects link between *Criticism* and *Issue*, issue and issue analysis nodes are brought into the perspective. The state nodes for these analysis are also selected.
- *Requirements*. We select *Requirements, Scenarios, Demonstrations,* and *Prototypes* to complete the picture.

The resulting perspective pattern is defined as follows:

$N(\pi)$ = *{User, poses, Criticism, CriticismAnalysis, initiates, hasState, StepState, affects, Issue, IssueAnalysis, RequirementsAnalysis, Requirement, Scenario, tests, uses, Demonstration, demonstrates, Prototype}*

One issue this example points out is that since links have classes themselves and they can appear connecting multiple types of nodes, selecting them once will indicate their presence in the pattern in all places where they maintain a weakly connected graph, even if not desired elsewhere in the pattern. Schema designers need to take care to only reuse the

same association name when the association is truly the same. This is the reason that new association names were introduced beyond those used by Ibrahim. Conceivably there will be times when a designer does not want to use the same name twice anywhere in the schema.

Given such a perspective pattern, a filter set for the *name* attribute of *User* equaling a particular string will limit the display to all criticisms posed by a particular user, the issues and requirements they relate to, and the prototypes, demonstrations and scenarios that show the solutions. The states of all the analyses initiated based on these artifacts. The user can then inspect the issues that were derived from the criticisms and check to see what versions of the system should contain solutions.

2. Manager Checks Ongoing Analysis Efforts

Another perspective can address a manager's need to see what analysts are assigned to what analysis efforts. In this case the perspective pattern is set by selecting *asssignedTo* and *hasState* as a minimum. This provides a pattern that appears as in the following sets:

$N(\pi)$ = *{Analyst, assignedTo, CriticismAnalysis, IssueAnalysis, RequirementsAnalysis, hasState, StepState}*

By filtering to only show busy = True analysts, the manager can see what work is currently being done.

VII. CONCLUSION AND DIRECTION FOR FUTURE RESEARCH

A. HOPE ARCHITECTURE

In this thesis, the architecture used for MORE [Lucarella and Zanzi, 1996] has been modified to work with a hypergraph instead of a graph model. All of the operations developed for MORE have been redesigned for hypergraphs in HOPE. As hypergraph models can be used to describe analysis patterns [Luqi, et. al., 1994] [Luqi and Goguen, 1997] this aids in the use of the hypermedia system in supporting analysis.

This extension of the model has also allowed a closer mapping to the Dexter Hypertext Reference Model [Halasz and Schwartz, 1994], which is intended to support application integration into hypermedia systems and hypermedia transportability. Dexter requires support for n-ary links and link-to-link connections. Support for composites was not addressed in HOPE and is another area where MORE falls short of the Dexter requirements. This is another area for follow-up research. Composites within the hypergraph model will essentially create another type of abstraction (in this case based on aggregation) for the model and displays. It is suspected that the mathematical model will need further evolution to support this capability.

Link integrity between the storage layer and the within content layer has not been adequately solved by any of the research reviewed for this thesis. This thesis does not provide a solution either. The problem is complicated for analysis systems in that a change to the contents of a node may invalidate a link made between components by an analyst. It is vital that link integrity solutions be explored with analysis systems in mind.

The method used to develop the hypergraph model has had the additional benefit of putting links on an equal footing with component nodes. Links can be abstracted, filtered, linked, searched for, and returned to run-time applications. This is important to analysis applications as the links represent the value added by the analyst.

Most hypermedia research systems have focused on designs for non-sequential authoring and reading. Some analysis support has been done for argumentation and software engineering. However, this is perhaps the first example of a framework for easily constructing hypermedia analysis support systems.

Finally, HOPE allows multiple hyperobject multimedia systems to reside within the same storage layer. This can be used to define private workspaces for analysts working particular problems. The results of their efforts should be merged with a HOMIS representing the general knowledge of the project, or multiple HOMIS should be able to be "virtually merged" indicating areas of overlap and connection. In order for an analyst to experiment with multiple solutions to a criticism, such private spaces need to be established. The answer actually selected needs to be brought back into the hypermedia structure.

B. PRESENTATION ISSUES

This effort extended some of the presentation options previously available through graph-based hypermedia models. Abstraction is used to improve the filtering ability of the user, and links are a legitimate focus of perspectives and filtering options. Previous research efforts [Nielson, 1995] [Marshall and Shipman, 1993, 1997] demonstrated that overview graphs of hypermedia systems enhance user understanding. The work in this thesis made use of that research as well as work that demonstrated methods to improve user comprehension of queries through visual representation [Consens and Mendelzon, 1989] [Lucarella and Zanzi, 1996]. What is not known is whether the use of abstraction will be comprehended by the user. Research should be done to see whether hiding information through the use of abstraction actually makes the job of the analyst become easier or more difficult.

Some other aspects need more research. In particular the positioning of nodes and links in a frame is made more difficult by the analysis use of the hypermedia. Positions cannot be stored with nodes as is done in other systems since modification of the hypermedia can be done by multiple users all working under different perspectives. With different perspectives set, the positioning of the nodes and links would need to be different for each of the users. Others are working on algorithms for automatically positioning nodes and edges of graphs. At least one of these need to be integrated with HOPE tools

for evaluation. The systems built using HOPE are not yet fully capable of being used with even modest amounts of information until such a capability is included.

Another area for future research is defining filters based on complex attributes. The filters defined above only take into account the values of attributes with primitive types. However, attributes connecting components (either directly or *mildly*) to components containing particular content could work. In addition, filters that are based on attributes connecting targeted structures of sub-hypergraphs could be defined as well.

LIST OF REFERENCES

Alexander, C., Ishikawa, S., Silverstein, M., Jacobsen, M., Fiksdahl-King, I., and Angel, S., *A Pattern Language*, Oxford University Press, New York, NY, 1977.

Arnold, K. and Gosling, J., *The Java Programming Language*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1996.

Awad, M., Kuusela, J., and Ziegler J., *Object-Oriented Technology for Real-Time Systems*, Prentice Hall PTR, Upper Saddle River, NJ, 1996.

Badr, S., and Luqi, "Automation Support for Concurrent Software Engineering", *Proceedings of the 6th International Conference on Software Engineering and Knowledge Engineering*, June 1994, pp. 46-53.

Berzins, V., Dampier, D., "Software Merge: Combining Changes to Decompositions", *Journal of Systems Integration*, Mar 1996, pp. 135-150.

Berzins, V. and Luqi, *Software Engineering with Abstraction*, Addison-Wesley, 1991.

Botafogo, R. and Shneiderman, B., "Identifying Aggregates in Hypertext Structures", *Third ACM Conference on Hypertext Proceedings*, San Antonio, Texas, 1991, pp. 63-74.

Bush, V., "As we may think", *Atlantic Monthly*, July 1945, pp. 101-108. Reproduced in hypertext at http://www.isg.sfu.ca/~duchier/misc/vbush/.

Casanova, M., Tucherman, L., Lima, M., Netto, R., Rodriguez, N., and Soares, L., "The Nested Context Model for Hyperdocuments", *Third ACM Conference on Hypertext Proceedings*, San Antonio, Texas, 1991, pp. 193- 201.

Coad, P., and Mayfield, M., *Java Design: Building Better Apps & Applets*", Yourdon Press, Upper Saddle River, NJ, 1996.

Conklin, J. and Begeman, M., "gIBIS: A Hypertext Tool for Team Design Deliberation", *Hypertext '87 Paper,* Chapel Hill, North Carolina, 1987, pp. 247 - 251.

Consens, M. and Mendelzon, A., "Expressing Structural Hypertext Queries in GraphLog", *Hypertext '89 Proceedings*, Pittsburgh, PA, 1989, pp. 269-292.

Davis, H., Hall, W., Heath, I., Hill, G., and Wilkins, R., "Towards an Integrated Information Environment With Open Hypermedia Systems", *Proceeding of the ACM Conference on Hypertext*, Milano, Italy, 1992, pp. 181-190.

Galitz, W., *User-Interface Screen Design*, QED Publishing Group, Wellesley, MA, 1993.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, Mass., 1995.

Garg, P., and Scacchi, W., "On Designing Intelligent Hypertext Systems for Information Management in Software Engineering", *Hypertext '87 Paper,* Chapel Hill, North Carolina, 1987, pp. 409 - 432.

Goose, S., Dale, J., Hall, W., and De Roure, D., "Microcosm TNG: A Distributed Architecture to Support Reflexive Hypermedia Applications", *Proceedings of Hypertext 97*, Southampton, UK, 1997, pp. 226-227.

Goose, S., Dale, J., Hill, G., De Roure D., and Hall, W., "An Architecture to Support an Open Distributed Hypermedia System", *ECS Research Journal*, University of Southampton, Nov. 1995. http://www.ecs.soton.ac.uk/rj95/mm/sg93r/journal.htm.

Gronbaek, K., and Trigg, R., "Design Issues for a Dexter-Based Hypermedia Systems", *Commun. ACM. 37*, 2 (Feb. 1994), pp. 41-49.

Haake, J., Neuwirth, C., and Streitz, N., "Coexistence and Transformation of Informal and Formal Structures: Requirements for More Flexible Hypermedia Systems", *European Conference on Hypermedia Technology Proceedings*, Edinburgh, Scotland, 1994, pp. 1-12.

Halasz, F., "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems", *Hypertext '87 Paper,* Chapel Hill, North Carolina, 1987, pp. 345 - 365.

Halasz, F., and Schwartz, M., "The Dexter Hypertext Reference Model", *Commun. ACM. 37*, 2 (Feb. 1994), pp. 30-39.

Ibrahim, O., "A Model And Decision Support Mechanism For Software Requirements Engineering", Ph.D. Dissertation, Computer Science Dept., Naval Postgraduate School, Monterey, CA, 1996.

Jackson, M., *Software Requirements & Specification: a lexicon of practice principles and prejudices*, ACM Press, New York, 1995.

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G., *Object-Oriented Software Engineering – A Use Case Driven Approach*, Addison Wesley, Reading, MA, 1992.

Jordan, D., Russell, D., Jensen, A., and Rogers, R., "Facilitating the Development of Representations in Hypertext with IDE", *Hypertext '89 Proceedings*, Pittsburgh, PA, 1989, pp. 93-104.

Leggett, J., and Schnase, J., "Viewing Dexter With Open Eyes", *Commun. ACM. 37*, 2 (Feb. 1994), pp. 77-86.

Lucarella, D., Parisotto, S., and Zanzi, A., "MORE: Multimedia Object Retrieval Environment", *Hypertext '93 Proceedings*, Seattle, WA, 1993, p. 39-50.

Lucarella, D., and Zanzi, A., "A Visual Retrieval Environment for Hypermedia Information Systems", *ACM Transactions on Information Systems*, Vol. 14, No. 1 (Jan 1996), pp. 3-29.

Luqi, "Software Evolution through Rapid Prototyping", *IEEE Computer*, May 1989, pp. 13-25.

Luqi, "A Graph Model for Software Evolution", *IEEE Transactions on Software Engineering*, Vol. 16, No. 8, 1990.

Luqi and Berzins, V., "Rapidly Prototyping Real-Time Systems", *IEEE Software*, Sep. 1988, pp. 25-36.

Luqi, Berzins, V., and Yeh, R., "A Prototyping Language for Real-Time Software", *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, October 1988, pp. 1409-1423.

Luqi and Goguen, J., "Formal Methods: Promises and Problems", *IEEE Software*, Jan. 1997, pp. 73-85.

Luqi, Goguen J., and Berzins V., "Formal Support for Software Evolution", *Proceedings of the 1994 Monterey Workshop on Software Evolution*, Naval Postgraduate School, Monterey, CA, 1994, pp. 13-21.

Luqi and Ketabchi, M., "A Computer Aided Prototyping System", *IEEE Software*, March 1988, pp. 66-72.

Malcolm, K., Poltrock, S., Schuler, D., "Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise", *Third ACM Conference on Hypertext Proceedings*, San Antonio, Texas, 1991, pp. 13-24.

Marmann, M., and Schlageter, G., "Towards a Better Support for Hypermedia Structuring: The HYDESIGN Model", *Proceeding of the ACM Conference on Hypertext*, Milano, Italy, 1992, pp. 232-241.

Marshall, C., and Shipman III, F., "Searching for the Missing Link: Discovering Implicit Structure in Spatial Hypertext", *Hypertext '93 Proceedings*, Seattle, WA, 1993, p. 217-230.

Marshall, C., and Shipman III, F., "Spatial Hypertext and the Practice of Information Triage", *Proceedings of Hypertext 97*, Southampton, UK, 1997, pp. 124-133.

Mayhew, D., *Principles and Guidelines in Software User Interface Design*, PTR Prentice Hall, Englewood Cliffs, NJ, 1992.

Nelson, T., *Literary Machines 93.1*, Mindful Press, Sausalito, CA, 1992.

Nielsen J., *Hypertext and Hypermedia*, Academic Press, San Diego, CA, 1990.

Nielsen, J., *Multimedia and Hypertext*, AP Professional, Cambridge, MA, 1995

Nurnberg, P., Leggett, J., and Schneider, E., "As We Should Have Thought", *Proceedings of Hypertext 97*, Southampton, UK, 1997, pp. 96-101.

Osterbye, K. and Normark, K., "An Interaction Engine for Rich Hypertexts", *European Conference on Hypermedia Technology Proceedings*, Edinburgh, Scotland, 1994, pp. 167-176.

Pearl, A., "Sun's Link Service: A Protocol for Open Linking", *Hypertext '89 Proceedings*, Pittsburgh, PA, 1989, pp. 137-146.

Pree, W., *Design Patterns for Object-Oriented Software Development*, ACM Press, Wokingham, England, 1995.

Rizk, A., and Sauter, L., "Multicard: An open hypermedia System", *Proceeding of the ACM Conference on Hypertext*, Milano, Italy, 1992, pp. 4-10.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

Schwabe, D., Rossi, G., and Barbosa, S., "Abstraction, Composition and Lay-Out Definition Mechanisms in OOHDM", *Electronic Proceedings of the ACM Workshop on Effective Abstractions in Multimedia*, San Francisco, CA, 1995, http://www.cs.tufts.edu/~isabel/schwabe/MainPage.html

Shneiderman, B. and Kearsley, G., *Hypertext Hands-On! An Introduction to a New Way of Organizing and Accessing Information*, Addison-Wesley Publishing Company, Reading, MA, 1989.

Streitz, N., Hannemann, J., and Thuring, M., "From Ideas and Arguments to Hyperdocuments: Travelling through Activity Spaces", *Hypertext '89 Proceedings*, Pittsburgh, PA, 1989, pp. 343-364.

Tochtermann, K. and Dittrich, G., "Fishing for Clarity in Hyperdocuments with Enhanced Fisheye-Views", *Proceeding of the ACM Conference on Hypertext*, Milano, Italy, 1992, pp. 212-221.

Travers, M., "A Visual Representation for Knowledge Structures", *Hypertext '89 Proceedings*, Pittsburgh, PA, 1989, pp. 147-158.

"UML Notation Guide", http://www.rational.com/uml/references/notation_guide.html, Version 1.0, 1997.

Will, U. and Leggett, J., "Hyperform: A Hypermedia System Development Environment", *ACM Trans. Inf. Syst.* 15, 1 (Jan. 1997), 1-31.

APPENDIX A. CAPS ACTORS

1. ACTOR SUMMARY

Actors are the human or autonomous agents that exist outside the automation boundary for CAPS. The following table summarizes information about CAPS actors. The terms used in the table relate to the type of interaction the actor has with CAPS tools and information.

Active/Passive refers to whether or not the actor initiates interaction with the tools and information. An active actor initiates the interaction while a passive actor responds when a request is forwarded only.

Client/Non-client refers to whether or not the actor is using the tools for a particular purpose, or if they merely affect the system.

Primary/Secondary indicates if the actor is one of the reasons for the system's existence. The existence of an administrator is not a reason to have CAPS tools, but the administrator could play an important role.

	Active/Passive	Client/Nonclient	Primary/Secondary
Project Manager	Active	Client	Primary
Requirements Analyst	Active	Client	Primary
Software Designer	Active	Client	Primary
Stakeholder	Active	Client	Primary
User	Active	Client	Primary
Administrator	Active	Client	Secondary
Analyst	Active	Client	Primary

Table A-1 Actor Summary

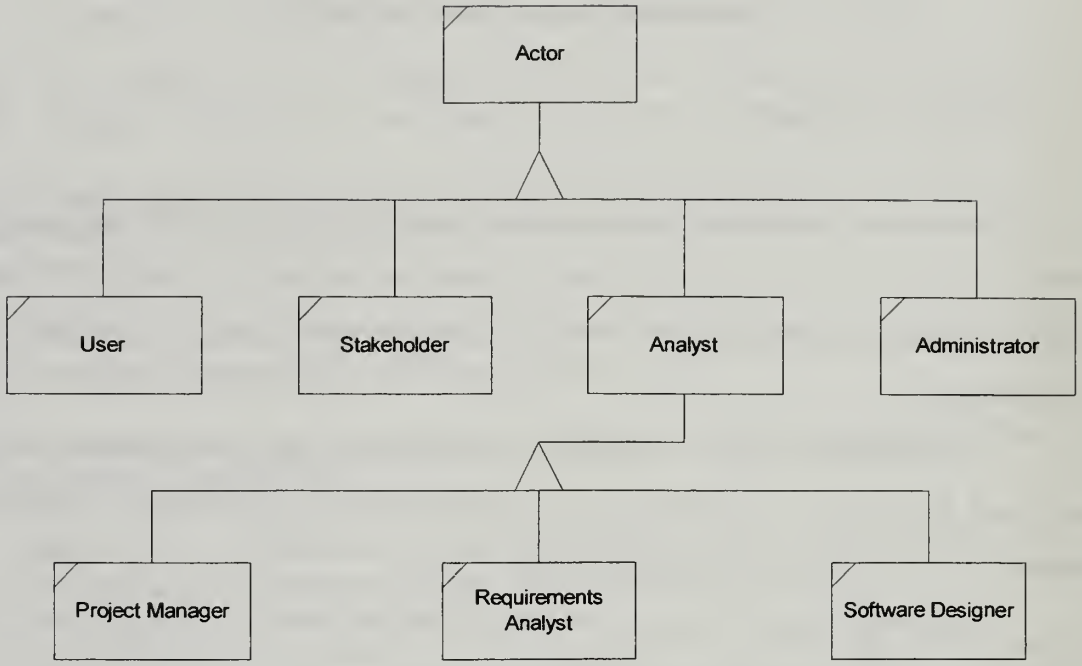

Figure A-1 Actor Hierarchy

2. ACTOR DESCRIPTIONS

Project Manager (PM) is responsible for the execution of the project. The PM is concerned with scheduling individuals to tasks, reviewing the status of the project, and choosing between alternative solutions offered for each issue.

Requirements Analyst (RA) is responsible for reviewing user comments and distilling out the issues to be resolved by system requirements. Alternative requirements or requirement changes are developed by RA for each issue.

Software Designer (SD) attempts to satisfy requirements either through changes to existing components or the development of new software components.

Stakeholder is an advisor to the project. Stakeholders effectively make up the projects board of directors. Stakeholders represent different interest groups within the project (e.g., users or sponsors).

User is an individual being asked to try a prototype system and provide feedback. In a more mature system or in alternate methods, this system does not need to be a

prototype and the user can be any user of the software. Users operate the system and provide criticisms back to the project.

Administrator takes care of the support software and database. Administrator sets up the schema, adds users to access lists, provides storage volumes to the database and performs other such tasks. The decision support system is not built for the administrator, but to make the system run smoothly and economically it must be built with the administrator's tasks in mind.

Analyst is a generalization of all analytical actors. These include PM, RA, and SD.

APPENDIX B. USE CASE ANALYSIS

The following use cases describe the requirements for the CAPS schema and run-time layer applications. The HOPE architecture is assumed and is therefore used within the requirements even though not a part of the target domain.

1. USE SUMMARY AND RELATIONSHIPS

	Uses	Tools Section(from Ch. IV)	Actor
U1	U2, U3, U4, U5	Open and close session, add/edit component nodes, link nodes, delete component and link nodes.	Project Manager
U2	None	openSession	Actor
U3	None	Add/edit component nodes and link nodes, delete component nodes and link nodes.	Analyst
U4	None	Realize edits	Analyst, Administrator
U5	U4	Close a session, realize edits	Actor
U6	U2, U3, U4, U5	Open and close session, add/edit component nodes, link nodes, delete component and link nodes.	Project Manager
U7	U2, U4, U8	Open and close session, Retrieve a component node content.	Software Designer
U8	None	Retrieve component node content.	Analyst
U9	None	None	Administrator
U10	U2, U3, U4, U5	Open and close session, add/edit component nodes, link nodes, delete component and link nodes.	Project Manager
U11	U8	Open and close session, add/edit component nodes, link nodes, delete component and link nodes, Retrieve a component node content.	User, Stakeholder
U12	U2, U3, U4, U5, U8	Open and close session, add/edit component nodes, link nodes, delete component and link nodes, retrieve a component node content.	Requirements Analyst
U13	U2, U3, U4, U5, U8, U14, U15	Open and close session, add/edit component nodes, link nodes, delete component and link nodes, retrieve a component node content, calculate distance, calculate complexity	Requirements Analyst
U14	None	Copy a component node with links in tact.	Analyst
U15	None	calculate distance, calculate complexity	Analyst
U16	None	Search for components and links	Actor
U17	None	View and filter the hypergraph	Actor

	Uses	Tools Section(from Ch. IV)	Actor
E1	None	Open a session	Actor, Administrator
E2	None	Retrieve a component's content	Actor

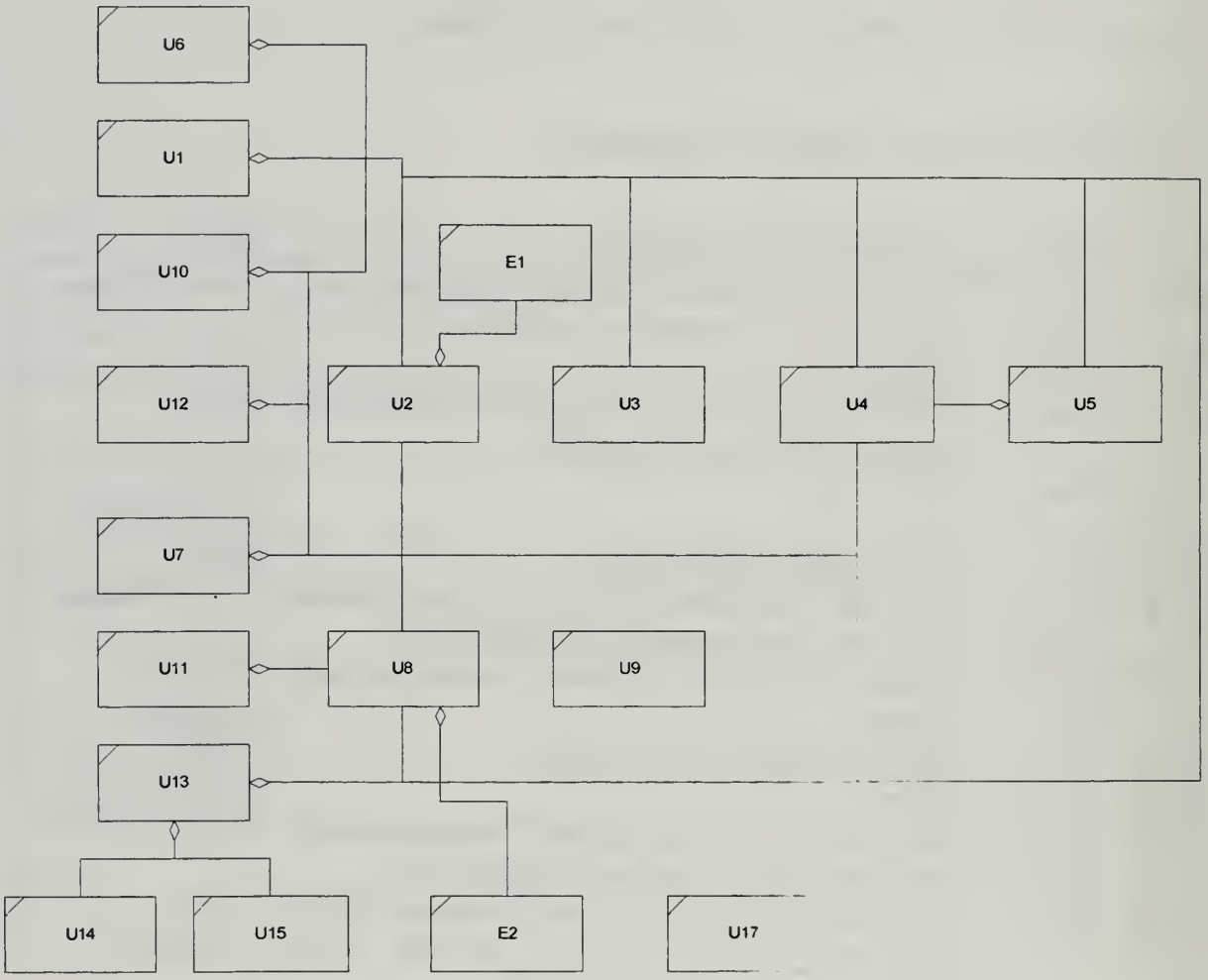

Figure B-1 Use Case Relationships

2. USE CASE SHEETS

Use Case	(U1) Record Initial Requirements
Actors	Project Manager (PM)
Preconditions	A HOMIS must be established within the HOPE storage layer. The HOMIS has a schema defined appropriate for CAPS and users defined to allow access to people in their appropriate roles.

Description	The PM opens a session selecting the HOMIS for this software engineering project. The PM modifies the HOMIS, by adding components representing the initial requirements for the system. Requirement nodes are linked together where uses relationships occur to create a hierarchy of requirements. The PM can save the HOMIS periodically during the session. Upon ending the session, the PM is prompted to save the changes to the hypermedia system and exit.
Sub Use Cases	(U2) Open a session, (U3) Modify the HOMIS, (U4) Save Edits (U5) Close a session
Exceptions	There is no HOMIS available, or the HOMIS is locked for edit already (note that further work into merging hypermedia is required to allow concurrent use).
Activities	
Postconditions	Same as preconditions.

Use Case	(U2) Open a Session
Actors	Actor
Preconditions	Users exist with appropriate permission to open a session.
Description	The actor requests a session with a HOMIS. A list of existing HOMIS are provided and the option to select one or create a new one providing a name for the HOMIS. The user selects the HOMIS for the project being analyzed. Actor is asked if this is to be an edit or read only session. If the user has appropriate permission, a session is opened.
Sub Use Cases	None
Exceptions	(E1) User does not have permission to open the selected session with the HOMIS desired. No session is opened if this is the case. Message is sent to Administrator.
Activities	none
Postconditions	A session is opened to the HOMIS selected.

Use Case	(U3) Modify the project database HOMIS
Actors	Analyst
Preconditions	Analyst has an open edit session with the required HOMIS.
Description	Analyst sees a hypergraph view of the project database. The analyst creates new component nodes representing criticisms,

	issues, requirements, or designs and links them to other nodes with which they are associated. Links are also made with nodes that already exist. Deletion of components and links are also done as part of the analysis.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	None

Use Case	(U4) Save Edits
Actors	Analyst, Administrator
Preconditions	Analyst has an open session with a HOMIS. Changes have been made.
Description	Analyst requests that edits be saved. System persists the new copy of the HOMIS
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	Persistent copy of the HOMIS now reflects state as currently viewed by the user. (Note later approach may include merging hypermedia versions to allow concurrent edits).

Use Case	(U5) Close a Session
Actors	Actor
Preconditions	Actor has an open session with a HOMIS.
Description	Actor requests that session be closed. Actor is given the choice to save edits if the session is an edit session. Edits are realized if necessary and the session is closed.
Sub Use Cases	(U4) Save Edits
Exceptions	None
Activities	None
Postconditions	Session is closed.

Use Case	(U6) Design Prototype
Actors	Project Manager

Preconditions	A HOMIS exists for the project database under consideration.
Description	The project manager allocates requirements to system design components by either linking requirements to existing components or creating new design components and linking them to the requirements. Design components may be linked to each other to provide uses relationships among the components.
Sub Use Cases	(U2) Open session, (U3) Modify HOMIS, (U4) Save Edits, (U5) Close session.
Exceptions	None
Activities	None
Postconditions	Session is closed.

Use Case	(U7) Design Software
Actors	Software Designer
Preconditions	A HOMIS exists for the project database under consideration. At least one design component has been generated and is related to at least one requirement (there is a path to a requirement).
Description	The software designer (SD) opens a read only session (this does not require a lock on the hypertext, it is merely used to retrieve a component for edit – the editor determines locking on the content) to the HOMIS and retrieves the software component for edit. By opening the node for edit, the PSDL editor is launched and the designer creates and modifies the design. The PSDL editor is used to store content changes.
Sub Use Cases	(U2) Open session, (U8) Retrieve a Component, (U5) Close session.
Exceptions	None
Activities	None
Postconditions	Session is closed.

Use Case	(U8) Retrieve a Component
Actors	Actor
Preconditions	A session is open with a HOMIS.
Description	The actor selects a component through one of many user interface options and chooses to retrieve the contents for either viewing or editing. The presentation specification of the component is used to launch an appropriate application to work with the content.
Sub Use Cases	None

Exceptions	(E2) Anchor link to content does not exist.
Activities	None
Postconditions	A session is open with a HOMIS.

Use Case	(U9) View Security Log
Actors	Administrator
Preconditions	None
Description	The administrator opens the log and is able to view all entries. No edit capability is given. The administrator my delete or copy the entire log.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	None

Use Case	(U10) Plan Demonstration
Actors	Project Manager
Preconditions	A HOMIS representing the project database is present in the HOPE.
Description	The project manager opens an edit session with the HOMIS. By adding a demonstration node to the hypergraph, the demonstration is created. The project manager then links design components, requirements, issues, or criticisms with the demonstration. Since paths exist from all design components all the way back to requirements and through all issues and criticisms addressed to the users, any user may determine what is being demonstrated. Demonstration scenarios are also linked to the node. These also refer back to requirements and therefore to the other nodes connected to them. When finished, the project manager saves the edits and closes the session.
Sub Use Cases	(U2) Open a session, (U3) Modify the HOMIS, (U4) Save Edits (U5) Close a session
Exceptions	None
Activities	None
Postconditions	A demonstration node exists.

Use Case	(U11) Submit Criticism
Actors	User and Stakeholder
Preconditions	A demonstration exists in the HOMIS.
Description	The user or stakeholder opens a session with the HOMIS. A new criticism component is created and the contents are modified to reflect the users criticism. The user/stakeholder links the criticism (this could be done automatically by the user interface) to a particular demonstration scenario. The criticism is automatically linked to the user by the runtime layer application making the appropriate requests of the storage layer. The changes are saved and the session is closed.
Sub Use Cases	(U2) Open a session, (U3) Modify the HOMIS, (U4) Save Edits (U5) Close a session, (U8) Retrieve a Component
Exceptions	None
Activities	None
Postconditions	A new criticism node now exists..

Use Case	(U12) Analyze a Criticism
Actors	Requirements Analyst
Preconditions	A criticism exists within the HOMIS representing the project database.
Description	The requirements analyst opens a session with the HOMIS. Retrieving the criticism and reading it, the analyst either creates and adds text to a new issue component node, then links it to the criticism, or links the criticism to an existing issue node. The changes are saved and the session is closed.
Sub Use Cases	(U2) Open a session, (U3) Modify the HOMIS, (U4) Save Edits (U5) Close a session, (U8) Retrieve a Component
Exceptions	None
Activities	None
Postconditions	A new criticism node now exists..

Use Case	(U13) Explore Alternatives
Actors	Requirements Analyst
Preconditions	An issue to be resolved exists within the HOMIS representing the project database.

Description	The requirements analyst opens a session with the HOMIS. Retrieving the issue and reading it, the analyst either creates and adds text to new requirement component nodes, then links them to the issue, or links the issue to existing requirement nodes. Requirement nodes can also be copied and edited creating new versions of these requirements. The other links of the copied requirement must also be copied to the new node. These requirements must have a relationship with the original requirement illustrating the update. Multiple alternative solution relationships can be set up between issues and collections of requirements. Distance and complexity calculations are performed on each alternative to help make a decision. When a decision is made one alternative relation is changed to a uses relation. The changes are saved and the session is closed.
Sub Use Cases	(U2) Open a session, (U3) Modify the HOMIS, (U4) Save Edits (U5) Close a session, (U8) Retrieve a Component, (U14) Copy a node with links intact, (U15) Calculate distance and complexity
Exceptions	None
Activities	None
Postconditions	A new criticism node now exists..

Use Case	(U14) Copy a Node with links intact
Actors	Analyst
Preconditions	A component node exists in the HOMIS. An edit session is open.
Description	The analyst selects a component node to be copied. A new node is placed in the hypergraph with links in place to all nodes linked by the original object. A link from the original to the copy is automatically created indicating that this is an updated version.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	A new node now exists..

Use Case	(U15) Calculate Distance and Complexity
Actors	Analyst
Preconditions	A component node exists in the HOMIS. An session is open.

Description	The analyst selects a component node to be analyzed. The distance from the selected node to all leaf nodes, where no mildly connected paths are followed, is calculated. The longest distance is returned. The total number of links directed out from the node selected to leaf nodes is counted (again, no mildly connected paths are followed). This number is returned as the complexity.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	None

Use Case	(U16) Search
Actors	Actor
Preconditions	A session is open.
Description	The user determines first if the result of the search should be component nodes or links. The actor is then queried for attribute and content values that can be used to match the targets. A set of objects is returned to the actor.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	None

Use Case	(U17) View and filter the hypergraph
Actors	Actor
Preconditions	A session is open
Description	The actor determines what types of nodes and what values of component and link nodes are of interest. For instance, a user may wish to only see criticisms and resulting issues that were posed by the user.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	None

3. EXCEPTIONS

Exception	(E1) Permission Denied
Actors	Exception
Preconditions	A request has been made to open a session to a HOMIS without adequate permission.
Description	The user is informed that the operation cannot proceed. A message is sent to the system administrator's log.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	No session is opened.

Exception	(E2) Anchor Link is Broken
Actors	Exception
Preconditions	A request has been made to open the content of a node. No valid anchor link exists.
Description	The user is informed of the lack of an anchor. The node is removed from the hypermedia system. A message is sent to the administrator.
Sub Use Cases	None
Exceptions	None
Activities	None
Postconditions	The offending node is no longer in the hypermedia system. This is true regardless of the type of session that is open.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Center for Naval Analysis 1
   4401 Ford Ave.
   Alexandria, VA 22302

4. Dr. Ted Lewis, Chairman, Code CS/L 1
   Computer Science Dept.
   Naval Postgraduate School
   Monterey, CA 93943-5101

5. Chief of Naval Research 1
   800 North Quincy St.
   Arlington, VA 22217

6. Dr. Luqi, Code CS/Lq 1
   Computer Science Dept.
   Naval Postgraduate School
   Monterey, CA 93943-5101

7. Dr. Marvin Langston 1
   1225 Jefferson Davis Highway
   Crystal Gateway 2, Suite 1500
   Arlington, VA 22202-4311

8. David Hislop 1
   U.S. Army Research Office
   PO Box 12211
   Research Triangle Park, NC 27709-2211

9. Capt. Talbot Manvel 1
   Naval Sea Systems
   2531 Jefferson Davis Highway
   Attn: TMS 378 Capt. Manvel
   Arlington, VA 22240-5150

10. CDR Michael McMahon 1
Naval Sea System Command
2531 Jefferson Davis Highway
Arlington, VA 22242-5160

11. Dr. Elizabeth Wald 1
Office of Naval Research
800 N. Quincy Street
ONR Code 311
Arlington, VA 22217-5660

12. Dr. Ralph Wachter 1
Office of Naval Research
800 N. Quincy Street
Code 311
Arlington, VA 22217-5660

13. Army Research Lab 1
115 O'Keefe Building
Attn: Mark Kendall
Atlanta, GA 30332-0862

14. National Science Foundation 1
Attn: Bill Barnes
Div. Computer & Computation Research
1800 G St. NW
Washington, D.C. 20550

15. National Science Foundation 1
Attn: Bill Agresty
4201 Wilson Blvd.
Arlington, VA 22230

16. Hon. John W. Douglass 1
Assistant Secretary of the Navy
(Research, Development and Acquisition)
Room E741
1000 Navy Pentagon
Washington, D.C., 20350-1000

17. Technical Library Branch 1
Naval Command, Control and Ocean Surveillance Center
RDT&E Division, Code D0274
San Diego, CA 92152-5001

18. Head, Command and Control Department 1
Naval Command, Control and Ocean Surveillance Center
RDT&E Division, Code D40
San Diego, CA 92152-5001

19. Head, Command and Intelligence Systems Division 1
Naval Command, Control and Ocean Surveillance Center
RDT&E Division, Code D42
San Diego, CA 92152-5001